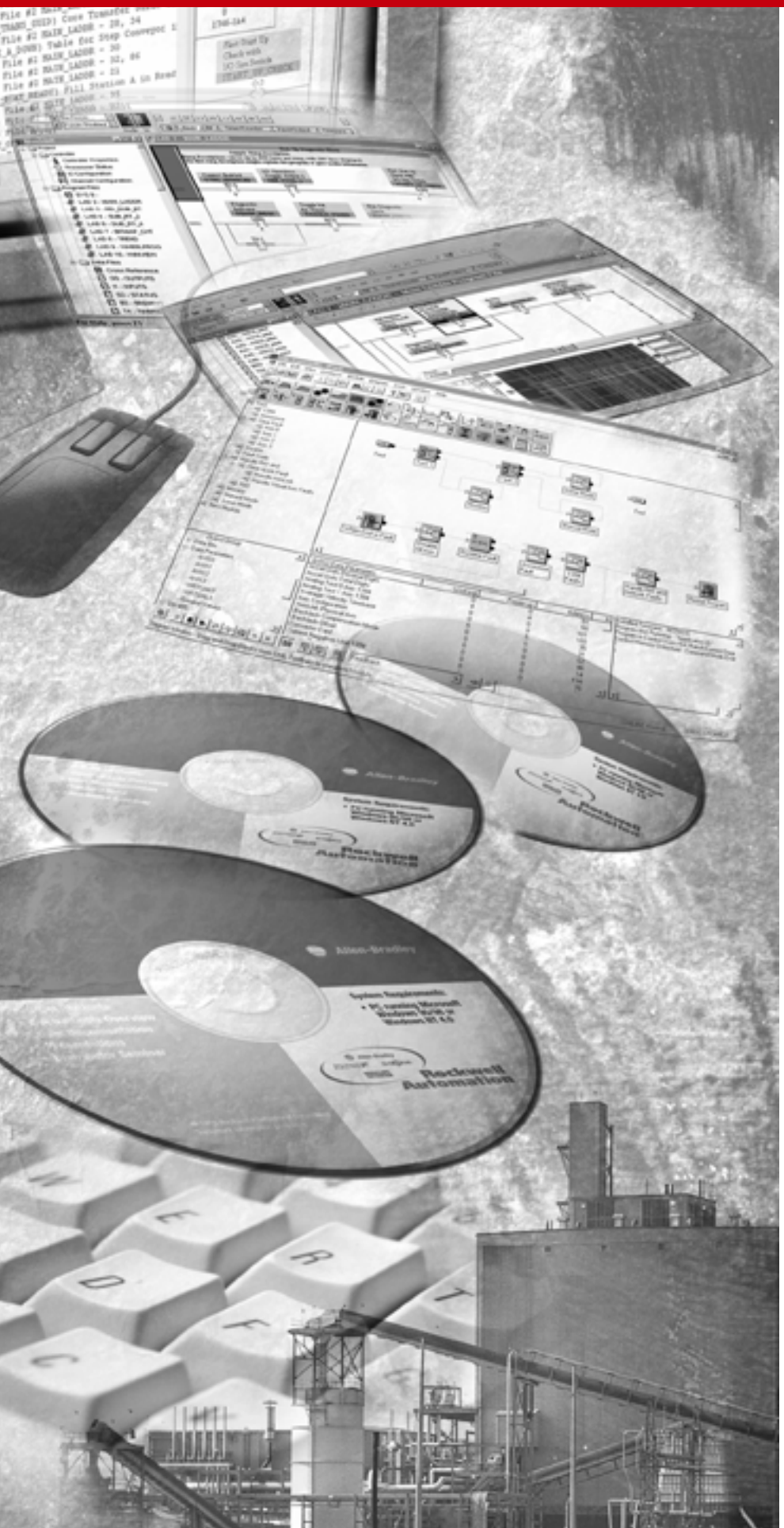


Allen-Bradley

Logix5000 Controllers Import/Export

1756 ControlLogix Controllers
1769 CompactLogix Controllers
1789 SoftLogix Controllers
1794 FlexLogix Controllers
PowerFlex 700s DriveLogix Controllers

Reference Manual

Rockwell Automation

Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI-1.1, *Safety Guidelines for the Application, Installation and Maintenance of Solid-State Control* (available from your local Allen-Bradley office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

Reproduction of the contents of this copyrighted publication, in whole or part, without written permission of Allen-Bradley Company, Inc., is prohibited.

Throughout this manual we use notes to make you aware of any safety considerations:

ATTENTION

Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss

Attention statements help you to:

- identify a hazard
- avoid a hazard
- recognize the consequences

IMPORTANT

Identifies information that is critical for successful application and understanding of the product.

Logix5550 is a trademark of Allen-Bradley, Inc., a Rockwell International Company.
Microsoft, Access, and Excel are trademarks of Microsoft.

Summary of Changes

This document describes how to use version 2.0 (major revision 2, minor revision 0) of the import/export utility that is included with RSLogix 5000 programming software, version 9.0.

Changes made for version 9.0 include:

- The AXIS tag was replaced with AXIS_CONSUMED, AXIS_SERVO, AXIS_SERVO_DRIVE, and AXIS_VIRTUAL tags. See pages 3-16 to 3-22.
- For any attribute that you can specify a “not applicable” state, you must enter <NA>, rather than just NA. See chapter 3.
- This revision of the manual includes a description and example of the STRING data type. See page 3-28.
- Appendix B now maintains a history of changes made to the import/export utility, starting with this version.

IMPORTANT Version 9 of RSLogix 5000 programming software only supports ControlLogix processors. You can only create new projects using catalog numbers 1756-L1, 1756-L53 and 1756-L55 processors. If you attempt to open or convert a FlexLogix, CompactLogix, SoftLogix, or DriveLogix project, an error message is displayed indicating these are unsupported processor types.

Notes:

Chapter 1

Importing and Exporting Files

Introduction 1-1
Importing a Complete Text File into a Project. 1-2
Exporting a Complete Project to a Text File 1-3
Importing a Tags Text File into a Project 1-4
Exporting Tags to a Text File. 1-6
Selecting the scope of the tags to export. 1-6

Chapter 2

Structuring a Complete (.L5K) Import/Export File Format

Introduction 2-1
Conventions 2-1
Internal file comments 2-1
Placing Information in an Import/Export File 2-2
Display style 2-3
Component descriptions. 2-3
Defining a Controller. 2-4
Specifying CONTROLLER attributes. 2-5
CONTROLLER guidelines 2-6
CONTROLLER example 2-6

Chapter 3

Creating a Complete Import/Export File

Introduction 3-1
Defining a Data Type 3-1
Specifying DATATYPE attributes. 3-2
Specifying a DATATYPE member 3-2
Specifying DATATYPE member attributes 3-4
DATATYPE guidelines 3-4
DATATYPE example 3-4
Defining a Module 3-5
Specifying MODULE attributes 3-5
Specifying a MODULE connection list. 3-7
Specifying MODULE connection list attributes. 3-8
MODULE guidelines. 3-8
MODULE example 3-8
Defining a Tag 3-10
Defining a TAG declaration for a non-alias tag 3-11
Defining a TAG declaration for an alias tag. 3-12
Defining an array specification within a TAG 3-12
Specifying TAG attributes. 3-13
Defining TAG initial values. 3-28
Defining a comment for a TAG component. 3-29
TAG guidelines 3-30
TAG examples. 3-30

Defining a Program 3-31
Specifying PROGRAM attributes 3-31
PROGRAM guidelines 3-32
PROGRAM example 3-32
Defining a Ladder Logic Routine 3-33
Specifying ROUTINE attributes 3-33
Defining a Function Block Diagram Routine 3-33
Specifying FBD_ROUTINE attributes 3-34
Defining a Task 3-34
Specifying TASK attributes 3-35
TASK guidelines 3-35
TASK example 3-36
Defining a Controller Object 3-36
Specifying CONFIG attributes 3-37
CONFIG examples 3-41

Chapter 4

Entering Ladder Diagram Logic

Introduction 4-1
Entering Rung Logic 4-1
Rung guidelines 4-2
Ladder ROUTINE example 4-2
Entering Branches 4-2
Example with a single branch 4-3
Example with two simultaneous branches 4-3
Entering Rung Comments 4-3
Entering Neutral Text for Ladder Instructions 4-3

Chapter 5

Entering Function Block Diagram Logic

Introduction 5-1
Entering Function Block Diagram Logic 5-1
SHEET guidelines 5-2
FBD_ROUTINE example 5-3
Entering IREFs and OREFs 5-5
IREF and OREF guidelines 5-6
IREF and OREF examples 5-6
Entering ICONs and OCONs 5-7
ICON and OCON guidelines 5-7
ICON and OCON examples 5-8
Entering Wires and Feedback Wires 5-8
WIRE guidelines 5-9
WIRE example 5-9
Entering Blocks 5-9
BLOCK guidelines 5-10
Entering Parameters for Function Block Instructions 5-11

Chapter 6

Structuring the Tag (.CSV) Import/Export File Format

Introduction 6-1
Conventions 6-1
    Internal file comments 6-2
Placing Information in a Tag (.CSV) Import/Export File . . . 6-2
Specifying a Tag Record 6-3
    Specifying dimensions 6-4
    TAG examples 6-4
Specifying an Alias Record 6-4
Specifying a Comment Record 6-5
Sample Scenarios of Importing/Exporting Tags. 6-6
Example Tag Import/Export File 6-7

Appendix A

Considerations for Using Microsoft Excel to Edit a .CSV File

Introduction A-1
Recommendations. A-1
RSLogix 5000 Data Transformations A-2
Microsoft Excel Data Transformation A-2

Appendix B

Import/Export Revision History

Introduction B-1
Backward Compatibility B-1
Motion Changes to Support the SERCOS Protocol. B-2
    MOTION_GROUP tag structure (version 1.1). B-3
    AXIS tag structure (version 1.1) B-3
Import/Export Version 1.1 RSLogix 5000 Version 8.0 . . . . . B-6

Notes:

Importing and Exporting Files

Introduction

This document describes how to use version 2.0 (major revision 2, minor revision 0) of the import/export utility that is included with RSLogix 5000 programming software, version 9.0.

With a Logix controller, you can import/export an entire project or you can import/export tags within a project.

When you import or export a project, you use the entire project. The text file is a complete import/export file, that includes tag definitions, data, program logic, I/O configuration information, and controller configuration information. If you import or export tags, the text file is a partial import/export file, that includes only tag definitions and tag comments.

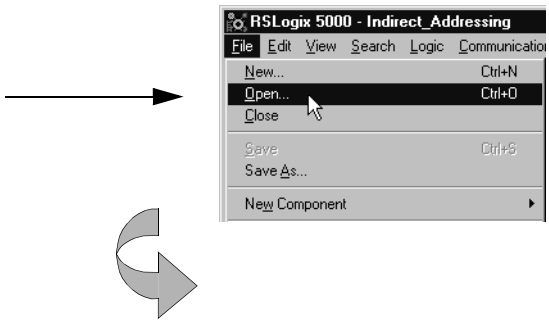
The structure of the import/export file depends on whether you perform a complete or partial import/export operation. There are also different considerations for complete and partial import/export operations. This chapter shows how to perform the import/export operations and describes any considerations.

When working with:	You can:	See Page:
projects	import a text file to create a project	1-2
	export a project into a text (.L5K) file	1-3
tags	import tags into a project	1-4
	export tags into a text (.CVS) file	1-6

Importing a Complete Text File into a Project

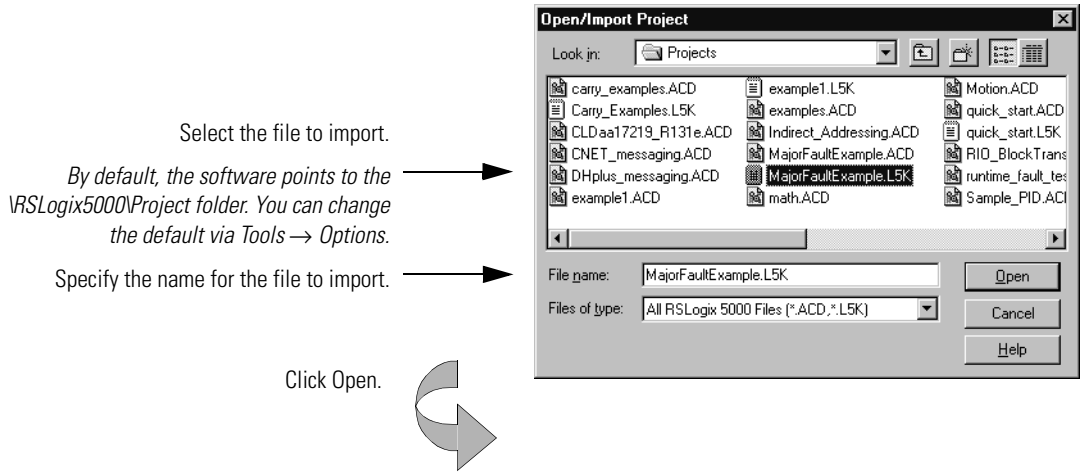
You can import controller information from a saved text file (that has a .L5K extension). This lets you use any text editor to create a project.

1. **Select File → Open.**

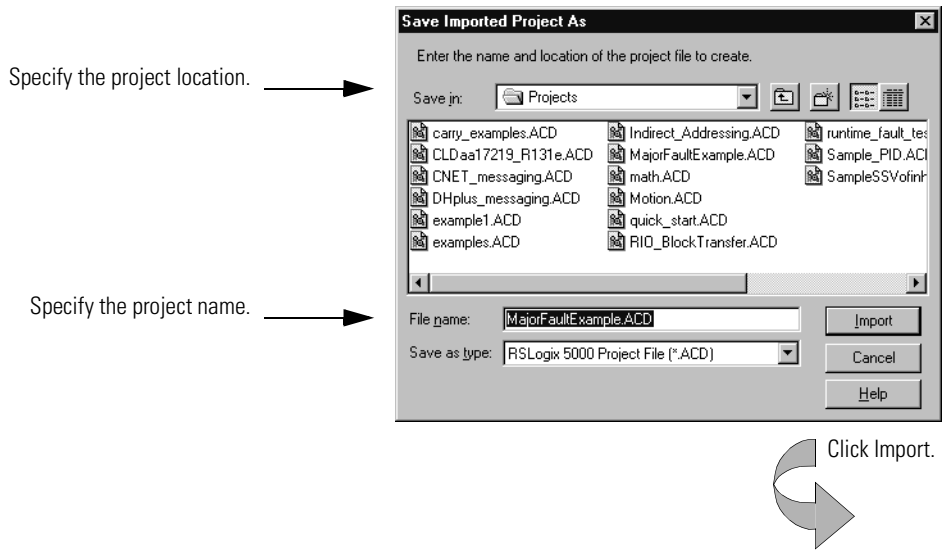


2. **Select the text file.**

The text file must have a .L5K extension.



3. **Specify the name and location of the project**



If you import a project that has forces, the project defaults to Forces Disabled, even if the project was exported with Forces Enabled.

For more information about the structure of the complete import/export file, see:

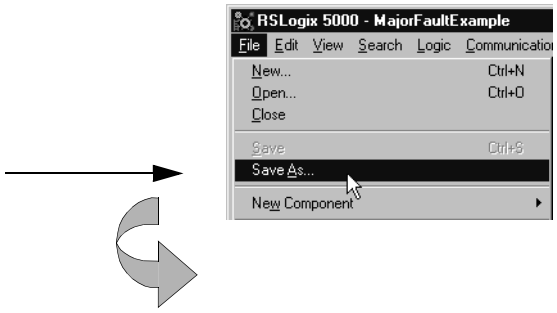
For information on:	See chapter:
structuring a complete import/export file	2
creating a complete import/export file	3
entering logic	4

Exporting a Complete Project to a Text File

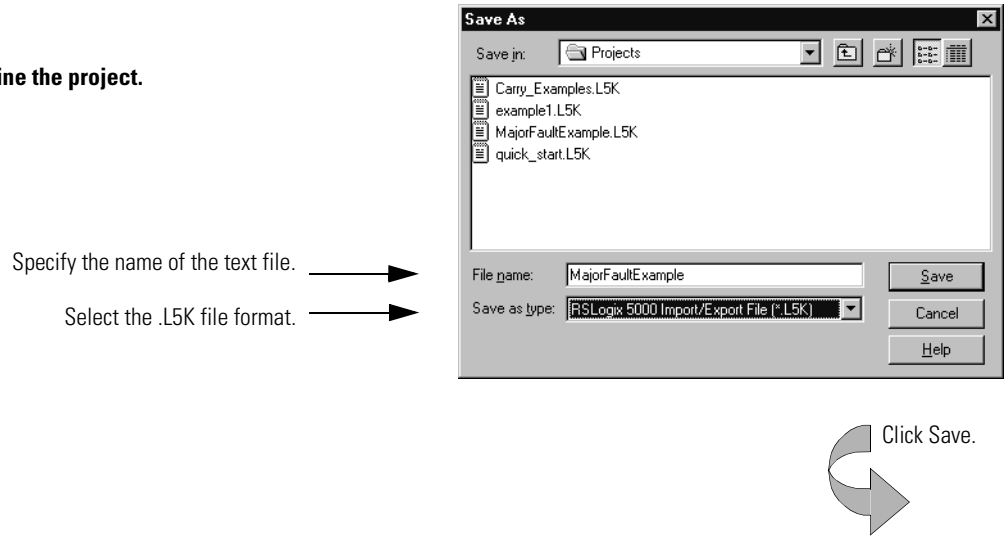
You can export the project to a text file. You can then use any text editor to modify the project.

Make sure the project you want to export is already open.

1. **Select File → Save As.**



2. **Define the project.**

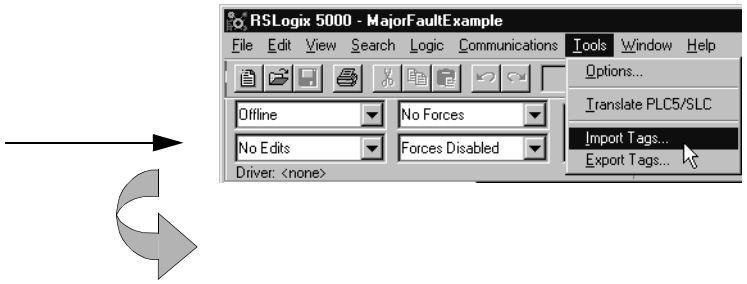


IMPORTANT Any unsaved edits are automatically saved when you OK the export operation.

Importing a Tags Text File into a Project

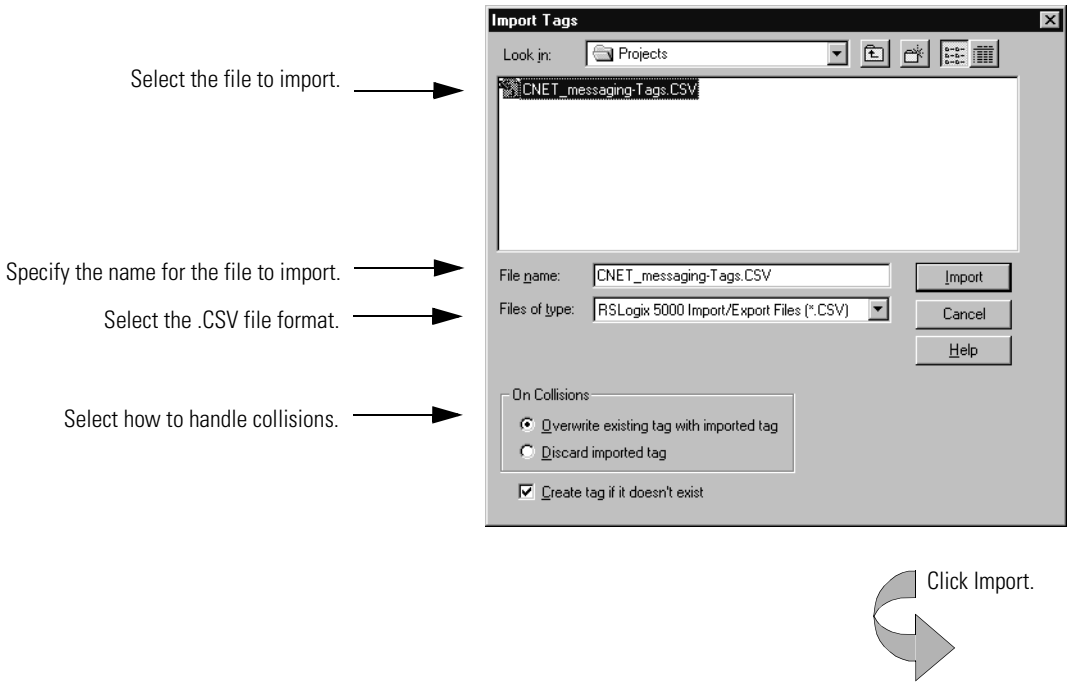
When you are offline and have a project open, you can import tags from a saved text file (that has a .CSV extension). This lets you use a database program (like Microsoft® Excel®) to create and edit tags.

1. **Select Tools → Import Tags.**



2. **Select the text file.**

The text file must have a .CSV extension.



When you import tags, the possibility exists for tags in the import file to have the same name as tags already in the open project. This condition is a collision. You specify how to handle a collision when you select the tag file to import:

If you want to:	Select:
replace the tag in the project with the tag from the import file	Overwrite (this is the default selection)
keep the tag that is in the project and discard the tag in the import file	Discard

It is also possible to have tags in the import file that do not exist in the open project. You can select whether to create these tags in the project.

If you delete tags from an import/export file and then import the file, tags are not deleted from the controller project. You have to use the programming software to delete tags from the tag list.

For more information about the structure of the partial import/export file, see:

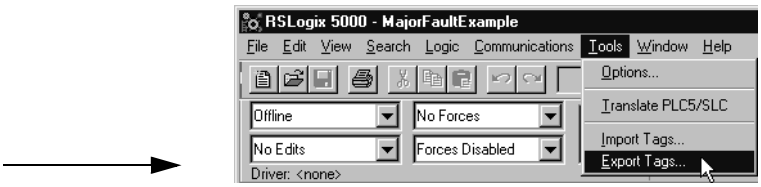
For information on:	See chapter:
structuring a partial import/export file	6

Exporting Tags to a Text File

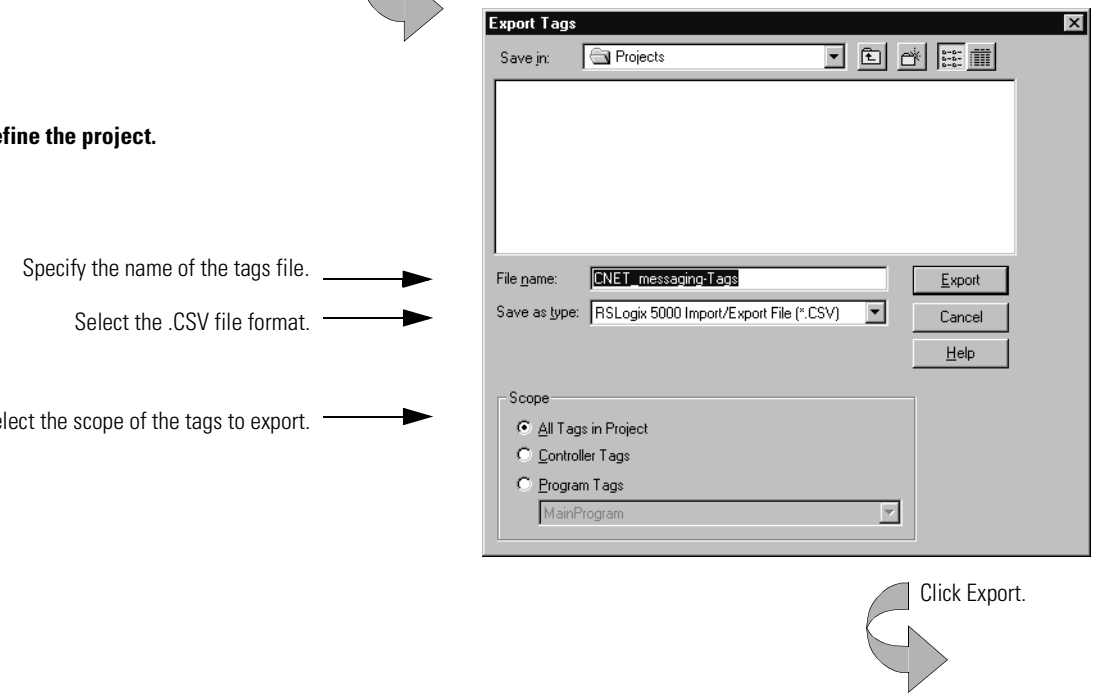
When you have a project open, you can export tags to a text file. You can then use a database program (like Microsoft Excel) to edit tags.

Make sure the project you want to export tags from is already open.

1. **Select Tools → Export Tags.**



2. **Define the project.**



Selecting the scope of the tags to export

When you export tags, you have these choices as to which tags in the project you want to export.

Scope:	This option exports:
All tags in project	all the tags (controller-scope and program-scope) in the project to a text file.
Controller tags	the controller-scoped tags of the project to a text file.
Program tags	the program-scoped tags of the program you specify. Use the drop-down arrow to display the available programs in the current project.

Structuring a Complete (.L5K) Import/Export File Format

Introduction

This chapter explains the overall structure of a complete import/export file. The file extension for a complete import/export file is .L5K.

For information about the specifics of each component in an import/export file, see the chapter "Creating an Import/Export File". For information on entering logic, see the chapter "Entering Logic."

Conventions

The import/export utility is based on the formats specified by the IEC 1131-3 specification. The examples follow these conventions:

Convention:	Meaning:
< >	items shown in angle brackets are required
[ ]	items shown in square brackets are optional
user_value	items in italics indicate user-supplied information
LITERAL	items in all uppercase indicate a required keyword or symbol that must be entered as shown
"["	items in double quotes are required characters

White space characters include spaces, tabs, carriage return, newline, and form feed. These characters can occur anywhere in an import/export file, except in keywords or names. If white space characters occur outside of descriptions, they are ignored.

Internal file comments

You can enter comments to document your import files. The import process ignores these comments. You can place comments anywhere in an import/export file, except in keywords, names, and component descriptions.

You can enter comments using either of these methods:

- Start the comment with two percent (%%) characters and stop at the end of the line.
- Start the comment with a "(*" and end with a corresponding "*)". Comments can extend multiple lines.

Placing Information in an Import/Export File

The import/export file contains different components of information. These components are:

Component:	Identifies:
CONTROLLER	name of the controller
DATATYPE	user-defined and I/O data structures
MODULE	modules in the controller organizer
TAG	controller-scope tags
PROGRAM	program files
ROUTINE	ladder logic routines
FBD_ROUTINE	function block diagram routines
TASK	controller tasks
CONFIG	configuration information

All components in an import export file follow this structure:

```
Component_Type <component_name> [Attributes]
        [body]
END_Component_Type
```

Where:

Item:	Identifies:
Component_Type	the component (as defined in above table)
component_name	a specific instance of the component
Attributes	any attributes of the component can also contain a description of the component separate each attribute with a comma (,)
body	any sub-components (children) of this component
END_Component_Type	end of the component information

Display style

Tags and data types support a radix attribute that lets you specify how to display the associated numerical information. The options are:

Display Option:	Example (based on 15 decimal):
Binary (uses a 2# prefix)	2#0000_0000_0000_1111
Octal (uses a 8# prefix)	8#000_017
Decimal	15
Hex (uses a 16# prefix)	16#000F
Ascii	'$00$0F'
Exponential	1.5000000e+01
Float	15.0

Component descriptions

Descriptions of components are optional. Unlike internal comments, descriptions are imported. Place the description within double quotes. For example:

```
TASK Task1 (Description := “Hello World”, Rate := 10000,
Priority := 10 )
END_TASK
```

To enter control characters in the description, precede the character with a dollar sign ($). The following table shows how to enter the supported control characters in a description:

For this character:	Enter:
$	$$
'	$'
"	$Q
10 (line feed)	$L or $l
13,10 (carriage return, line feed)	$N or $n
12 (form feed)	$P or $p
13 (carriage return)	$R or $r
9 (tab)	$T or $t
xxxx (4-digit character code that represents a hexadecimal value)	$xxxx

Defining a Controller

The CONTROLLER component is the overall structure of a project to be executed on one controller. It contains the configuration information and logic that you download to one controller. Preceding the CONTROLLER component, you have the header remarks (optional) and the version statement:

```
Import-Export
Version  := RSLogix 5000 - 8.00.00.00
Owner    := User Name, Allen-Bradley Co., Inc.
Exported := Fri Aug 24 10:25:38 2001

IE_VER := 2.0;
```

Following the header and version statement, the CONTROLLER component follows this structure:

```
CONTROLLER <controller_name>  [Attributes]
       [<DATATYPE declaration>]
       [<MODULE declaration>]
       [<TAG declaration>]
       [<PROGRAM declaration>]
       [<TASK declaration>]
       [<CONFIG controller objects declaration>]
END_CONTROLLER
```

Where:

Item:	Identifies:
controller_name	the controller name for the project
Attributes	attributes of the controller can also contain a description of the controller separate each attribute with a comma (,)
DATATYPE	I/O and user-defined data structures See page 3-1.
MODULE	devices in the controller organizer See page 3-5.
TAG	controller-scope tags See page 3-10.
PROGRAM	organization of routines See page 3-31.
TASK	organization of programs See page 3-33.
CONFIG	characteristics of controller objects (status information) See page 3-36.

Specifying CONTROLLER attributes

You can specify these attributes for a CONTROLLER:

Attribute:	Description:
`Description`	Provide information about the controller. Specify: `Description := "text"`
`ProcessorType`	Specify the type of controller (1756-L1, 1756-L53/A, 1756-L55/A, 1769-L20, 1769-L30, 1789-L60/A, 1794-L33/A, 1794-L34/A, PowerFlex700s) Specify: `ProcessorType := name`
`Major`	Specify the major revision number (1-127) of the controller. Specify: `Major := number`
`TimeSlice`	Percentage of available CPU time (10-90) that is assigned to communications. Specify: `TimeSlice := value`
`PowerLossProgram`	Name of the program to be executed on reboot after a power loss. Specify: `PowerLossProgram := name`
`MajorFaultProgram`	Name of the program to be executed when a major fault occurs. Specify: `MajorFaultProgram := name`
`CommPath`	List of the devices in the communication path. The communication path ends with the controller (`\Backplane\1`). Specify: `CommPath := device\device\device...\Backplane\1`
`RedundancyEnabled`	Specify whether redundancy is used or not. Enter a 0 to disable redundancy; enter a 1 to enable redundancy. Specify: `RedundancyEnabled := number`
`KeepTestEditsOnSwitchOver`	Specify whether to keep test edits on when a switchover occurs (only in a redundant system). Enter a 0 not to keep test edits on; enter a 1 to keep test edits on. Specify: `KeepTestEditsOnSwitchOver := number`
`DataTablePadPercentage`	Specify the amount of data table to reserve. If redundancy is not enabled, enter 0. If redundancy is enabled, enter 50. Specify: `DataTablePadPercentage := name`

CONTROLLER guidelines

Keep these guidelines in mind when defining a data type:

- All declarations must be explicitly ordered as shown in the syntax above.
- The maximum number of tasks depends on the controller type:

Controller:	Maximum Number of Tasks:
ControlLogix	32
SoftLogix5800	32
FlexLogix	8
CompactLogix (L20, L30)	4
DriveLogix	4

- There can only be one continuous task.
- Programs can only be scheduled under one task
- Scheduled programs must be defined - i.e. must exist

CONTROLLER example

```
CONTROLLER TestImportExport (Description := “Example",
TimeSlice := 11, MajorFaultProgram := Prg2)
END_CONTROLLER
```

Creating a Complete Import/Export File

Introduction

This chapter explains how to enter project and configuration information in a complete import/export file.

For information about:	See page:
Defining a data type	3-1
Defining a module	3-5
Defining a tag	3-10
Defining a program	3-31
Defining a routine	3-33
Defining a task	3-34
Defining a controller object	3-36

For information on entering logic, see the next chapter.

Defining a Data Type

A DATATYPE component follows this structure:

```
DATATYPE <DataType_name> [Attributes]
        [member_definition]
END_DATATYPE
```

Where:

Item:	Identifies:
`DataType_name`	the data structure
`Attributes`	attributes of the data structure can also contain a description of the component separate each attribute with a comma (,)
`member_definition`	each member of the data structure

Specifying DATATYPE attributes

You can specify these attributes for a DATATYPE:

Attribute:	Description:
`Description`	Provide information about the data type. Specify: `Description := "text"`
`Radix`	Specify the display style as decimal, hex, octal, binary, exponential, float, or ASCII. Specify: `Radix := value`
`Hidden`	Make the member a hidden member of the structure. Specify: `Hidden := 1`

Specifying a DATATYPE member

There are two kinds of data type members. A bit member is a member in which only a single bit of information is to be accessed. A non-bit member is a member that is defined as another data type (such as SINT, INT, DINT, COUNTER, etc.).

A non-bit member definition follows this structure:

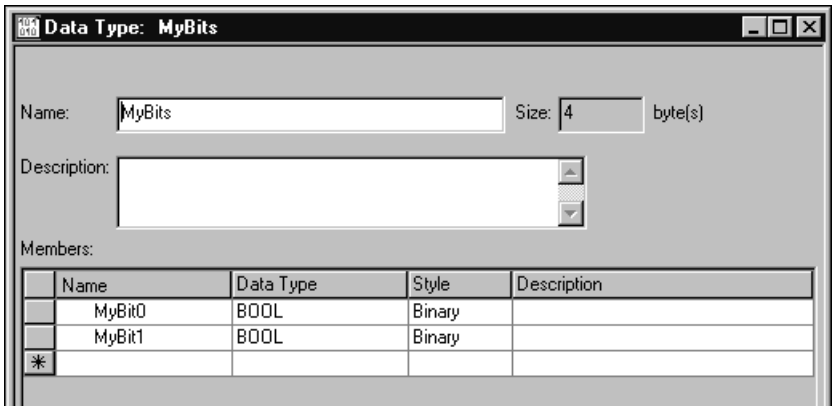
```
<TypeName> <MemberName> [Attributes];
```

All data types are allocated in 8-bit boundaries. A single bit of storage is not allowed, so a member cannot be a BOOL data type. To access a single bit, use the BIT declaration. BIT allows access to a single bit within a host member (a non-bit member).

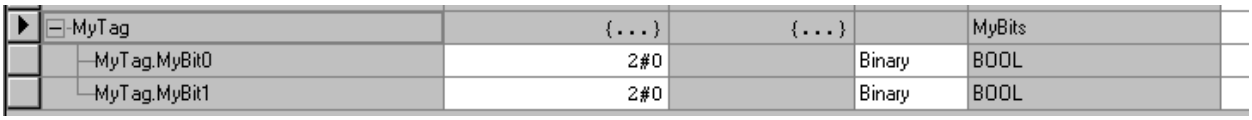
```
A bit member uses the following syntax:
BIT <BitName> <HostMemberName> : <BitPosition> [Attributes];
```

For example, create a user-defined datatype called “MyBits” and a tag called “MyTag” of type “MyBits.”

User-defined datatype “MyBits”



Tag “MyTag” of type “MyBits”



`ZZZZZZZZZZMyBits0` is the host member of MyBit0 and MyBit1. The datatype syntax for this example is:

```
DATATYPE MyBits (FamilyType := NoFamily)
       SINT ZZZZZZZZZZMyBits0 (Hidden := 1);
       BIT MyBit0 ZZZZZZZZZZMyBits0 : 0 (Radix := Binary);
       BIT MyBit1 ZZZZZZZZZZMyBits0 : 1 (Radix := Binary;
END_DATATYPE
```

The host member is normally a hidden member because only the bit references are visible when you define a tag of the datatype.

IMPORTANT There **must** be a space between the host member name and the colon and the colon and the bit position because type names can contain a colon (for example, I/O structures) and without the space we could not tell where type name actually ends.

Bit members cannot be defined before their host members. Note that BitPosition zero is the least significant bit.

Specifying DATATYPE member attributes

You can specify these attributes for a member of a DATATYPE:

Attribute:	Description:
`Description`	Provide information about the data type member. Specify: `Description := "text"`
`Radix`	Specify decimal, hex, octal, binary, exponential, float, or ASCII. Specify: `Radix := value`
`Hidden`	Make the member a hidden member of the structure. Specify: `Hidden := 1`

DATATYPE guidelines

Keep these guidelines in mind when defining a data type:

- Data types must be defined first within the controller body.
- Data types can be defined out of order. For example, if Type1 depends on Type2, Type2 can be defined first.
- Data types can be unverified. For example if Type1 depends on Type2 and Type2 is never defined, then Type1 will be accessible as an unverified type. Type2 will be typeless type. Tags of Type1 may be created but not of Type2.
- Data type members can be arrays but only one dimension is allowed.
- The following data types cannot be used in a user-defined data type: AXIS types, MOTION_GROUP, and MESSAGE.

DATATYPE example

```
DATATYPE MyStructure (FamilyType := NoFamily)
       DINT x;
       TIMER y[3] (Radix := Decimal);
       SINT MyFlags (Hidden :=1);
       BIT aBit0 MyFlags : 0 (Radix := Binary);
       BIT aBit1 MyFlags : 1 (Radix := Binary);
END_DATATYPE
```

Defining a Module

A MODULE component follows this structure:

```
MODULE <device_name> [Attributes]
       [ConfigData := <initial_value>;]
       [connection_list]
END_MODULE
```

Where:

Item:	Identifies:
`device_name`	the module
`Attributes`	attributes of the module can also contain a description of the module separate each attribute with a comma (,)
`ConfigData`	operating characteristics of the module
`connection_list`	connection characteristics for the module see page 3-7

Specifying MODULE attributes

You can specify these attributes for a MODULE:

Attribute:	Description:
`Description`	Provide information about the module. Specify: `Description := "text"`
`Parent`	If this module is a child to another module, specify the name of the parent module. The parent module must be defined before any child module. Specify: `Parent := name`
`CatalogNumber`	Specify the catalog number of the module. Specify: `CatalogNumber := number`
`Major`	Specify the major revision number (1-127) of the module. Specify: `Major := number`
`Minor`	Specify the minor revision number (1-255) of the module. Specify: `Minor := number`
`PortLabel`	Specify the port used to reach this module. The port label is either `RxBACKPLANE` for modules in a chassis or a text string for modules on a network. Specify: `PortLabel := label`
`ChassisSize`	Specify the number of slots in the chassis (1-32). This only applies to the MODULE statement that defines the controller selected for the project. Specify: `ChassisSize := number`
`Slot`	Specify the slot number (1-31) where the module is in the chassis. Specify: `Slot := number`

Attribute:	Description:
`NodeAddress`	Specify the ControlNet node address (1-99) or the remote I/O rack address (0-63) of the module. Specify: `NodeAddress := number`
`Group`	If the module is a remote I/O module, specify the starting group (0-7). For a block-transfer module, this is the module group number under the remote I/O adapter. Specify: `Group := number`
`CommMethod`	Specify the method of connecting to the module. Specify: `CommMethod := number`
`ConfigMethod`	Specify the method of configuring the module. Specify: `ConfigMethod := number`
`Mode`	Select a specific mode by setting the appropriate bit. **Set:** **For:** 0 fault in the module causes major fault in controller 2 inhibit the module Specify: `Mode := number`
`CompatibleModule`	Specify whether to connect to a compatible module based on the minor revision (*value* = 1) or to an exact match of the module (*value* = 0). Specify: `CompatibleModule := value`
`KeyMask`	Specify whether to connect to the exact module that matches the electronic keying information (vendor, product code, product type, major revision, minor revision). No keying will connect to any module. Specify: `KeyMask := hex_string`
`PrimCxnInputSize`	Specify the size of the data associated with the primary input connection (0-500 bytes). Specify: `PrimCxnInputSize := number`
`PrimCxnOutputSize`	Specify the size of the data associated with the primary output connection (0-496 bytes). Specify: `PrimCxnOutputSize := number`
`SecCxnInputSize`	Specify the size of the data associated with the secondary input connection (0-104 bytes). Typically, there is one I/O connection on a module (primary connection). If there are two, the second connection is the secondary connection. Specify: `SecCxnInputSize := number`
`SecCxnOutputSize`	Specify the size of the data associated with the secondary input connection (always 0). Typically, there is one I/O connection on a module (primary connection). If there are two, the second connection is the secondary connection. Specify: `SecCxnOutputSize := number`
`ChABaud`	For a 1756-DHRIO module, specify the baud rate for channel A. Enter 57.6, 115.2, or 230.4. Specify: `ChABaud := baud`
`ChBBaud`	For a 1756-DHRIO module, specify the baud rate for channel B. Enter 57.6, 115.2, or 230.4. Specify: `ChBBaud := baud`

Specifying a MODULE connection list

You can specify these attributes for a connection list:

```
CONNECTION <connection_name> [Attributes]
       [InputData := <value_list>;]
       [InputForceData := <value_list>;]
       [OutputData := <value_list>;]
       [OutputForceData := <value_list>;]
END_CONNECTION
```

Where:

Item:	Identifies:
`connection_name`	the connection
`InputData`	input channel data
`InputForceData`	forcing information for the input channel
`OutputData`	output channel data
`OutputForceData`	forcing information for the output channel
`Attributes`	attributes of the connection can also contain a description of the module separate each attribute with a comma (,)

For details on the data in the connection list, see the user manual for the I/O module. The connection list data depends on the I/O module and the configuration for that module.

Forces appear as arrays of bytes under the InputForceData and OutputForceData attributes of the connection list. Do not modify forces in the import/export file. Use the programming software to enter and enable forces.

Specifying MODULE connection list attributes

You can specify these attributes for a MODULE connection list:

Attribute:	Description:
`Description`	Provide information about the connection list. Specify: `Description := "text"`
`Rate`	Specify the requested packet interval (RPI) rate in microseconds. Specify: `Rate := microseconds`
`InputCxnPoint`	Specify the input connection point for the primary connection (0-255). Specify: `InputCxnPoint := number`
`OutputCxnPoint`	Specify the output connection point for the primary connection (0-255). Specify: `OutputCxnPoint := number`
`EventID`	Exists for future support. For now: Specify: `EventID := <NA>`

MODULE guidelines

Keep these guidelines in mind when defining a module:

- Attributes must be explicitly ordered as shown in the example on page 3-5.
- A parent module must be defined before any definitions of its child modules.

MODULE example

```
MODULE Local (Parent := Local,
       CatalogNumber := 1756-L1,
       Major := 1,
       PortLabel := RxBACKPLANE,
       ChassisSize := 10,
       Slot := 3,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111)

END_MODULE
```

```
MODULE DHRIO_Module (Parent := Local,
       CatalogNumber := 1756-DHRIO,
       Major := 2,
       PortLabel := RxBACKPLANE,
       Slot := 8,
       CommMethod := Standard,
       ConfigMethod := ChannelA RIO ChannelB DH,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111,
       ChABaud := 115.2,
       ChBBaud := 57.6)

       CONNECTION Standard (Rate := 500000,
              EventID := <NA>)
       END_CONNECTION
END_MODULE
MODULE Diagnostic_Module_1 (Parent := Local,
       CatalogNumber := 1756-OB16D,
       Major := 1,
       PortLabel := RxBACKPLANE,
       Slot := 5,
       CommMethod := Full Diagnostics - Output Data,
       ConfigMethod := Diagnostic,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111)
       ConfigData :=
[44,19,1,0,0,0,0,0,0,0,65535,65535,65535,0];

       CONNECTION Diagnostic (Rate := 5000,
              EventID := <NA>)
       END_CONNECTION

END_MODULE
```

```
MODULE input_1 (Parent := Local,
                        CatalogNumber := 1756-IA16,
                        Major := 2,
                        Minor := 1,
                        PortLabel := RxBACKPLANE,
                        Slot := 1,
                        CommMethod := 536870913,
                        ConfigMethod := 8388610,
                        Mode := 2#0000_0000_0000_0000,
                        CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
                        KeyMask := 2#0000_0000_0001_1111)
              ConfigData  :=
[28,16,1,0,0,0,1,9,1,9,0,0,0,0,65535,65535];
              CONNECTION StandardInput (Rate := 5000,
                                        EventID := 0)
              InputData  := [0,0];
              InputForceData :=
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,0,0,0,0,0,0,0,16,0];
              END_CONNECTION

       END_MODULE
```

Defining a Tag

You can define controller-scope tags and program-scope tags. Controller-scope tags are defined in one TAG component within the CONTROLLER component; program-scope tags are defined in a TAG component within a PROGRAM component within a CONTROLLER component. For example, all the tags for one program are defined in one TAG component within that PROGRAM component. A TAG component follows this structure:

```
TAG
       [tag_declarations]
END_TAG
```

Within a tag list, message and motion tags must follow all non-motion tags and axis tags must follow motion group tags.

IMPORTANT For detailed information about atomic and structure tags and their supported attributes and ranges, see the *Logix5000 Controllers Common Procedures Programming Manual*, publication 1756-PM001.

Defining a TAG declaration for a non-alias tag

A tag declaration for a non-alias tag follows this structure:

```
<tag_name> : <type[array_specification]> [Attributes][:= <initial_value>]
[, <tag_force_data>];
```

Where:

Item:	Identifies:
`tag_name`	name of the tag
`type`	type of tag atomic types: BOOL, SINT, INT, DINT, REAL string types: STRING predefined types: AXIS_CONSUMED, AXIS_SERVO, AXIS_SERVO_DRIVE, AXIS_VIRTUAL, CAM, CAM_PROFILE, CONTROL, COUNTER, MESSAGE, MOTION_GROUP, MOTION_INSTRUCTION, OUTPUT_CAM, OUTPUT_COMPENSATION, PID, SERIAL_PORT_CONTROL, TIMER function block types: unique type for each function block
`array`	dimensional boundaries for array tags see page 3-12
`Attributes`	attributes of the tag can also contain a description of the tag separate each attribute with a comma (,) see page 3-13
`initial_value`	initial value of the tag see page 3-28
`tag_forced_data`	list of forced tag data for an example, see page 3-30

There **cannot** be any whitespace between the type and array definition. There **must** be a space between the tag name and the colon and another space between that same colon and the type name. This is because type names can contain a colon and without the space it would be impossible to detect where the type name actually starts.

Defining a TAG declaration for an alias tag

A tag declaration for an alias tag follows this structure:

```
<tag_name> OF <alias> [Attributes];
```

Where:

Item:	Identifies:
`tag_name`	name of the alias tag
`alias`	name of the base tag the alias tag references Specify: *`alias<specifier>`* Where the *specifier* is: a bit (.*bitnumber*), array element ([*element*]), or structure member (.*membername*) of the tag.
`Attributes`	attributes of the tag can also contain a description of the tag separate each attribute with a comma (,)

Defining an array specification within a TAG declaration

An array specification follows this structure:

```
"["<element> [,<element> [,<element>] ]"]"
```

Where:

Item:	Identifies:
`element`	the number of elements within the array dimension for example: [5, 10, 2]

Specifying TAG attributes

You can specify these attributes for a TAG:

Attribute:	Description:
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Comment`	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`Radix`	Specify the display style as decimal, hex, octal, binary, exponential, float, or ASCII. Specify: `Radix := value`
`ProduceCount`	Specify the number of consumers allowed (any positive number). Specify: `ProduceCount := value`
`PLCMappingFile`	If this tag is mapped to a PLC controller, specify the file number (any positive number). Specify: `PLCMappingFile := number`
`PLC2Mapping`	If this tag is mapped to a PLC-2 file, set this attribute to 1. If this tag is not mapped to a PLC-2 file, set this attribute to 0. Specify: `PLC2Mapping := 1`
`Producer`	If the controller produces this tag, specify the name of the remote controller that consumes this tag. You must also specify a RemoteTag and RPI attribute. Specify: `Producer:= name`
`RemoteTag`	If the controller produces this tag to a controller that supports tag names, specify the name of the tag on the remote controller. You must also specify a Producer and RPI attribute. Specify: `RemoteTag := name`
`RemoteFile`	If the controller produces this tag to a PLC-5 controller, specify the PLC-5 file number (any positive number) on the PLC-5 controller. You must also specify a Producer and a RPI attribute. Specify: `RemoteFile := number`
`RPI`	If the controller produces this tag, specify the RPI value in milliseconds (any positive number). You must also specify a Producer and RemoteTag attribute. Specify: `RPI := milliseconds`

IMPORTANT If consume information is provided on an alias tag, the alias tag is converted to a base tag before it can consume data.

Specifying attributes for a MOTION_GROUP tag

A MOTION_GROUP tag differs from a basic tag. A motion group tag has these attributes:

Attribute:	Description:
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Comment`	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`GroupType`	Specify the type of motion group. Enter Warning Enabled or Warning Disabled. Specify: `GroupType := text`
`CourseUpdatePeriod`	Specify the coarse update period in milliseconds (1000-3200ms). Specify: `CourseUpdatePeriod := value`
`PhaseShift`	Specify the phase shift (0-65,535). Specify: `PhaseShift := value`
`GeneralFaultType`	Specify whether an error generates a major fault or a non-major fault. Enter Major Fault or Non Major Fault. Specify: `GeneralFaultType := text`
`AutoTagUpdate`	Enter Disabled or Enabled. Specify: `AutoTagUpdate := text`

Specifying attributes for a MESSAGE tag

A message tag differs from a basic tag. A MESSAGE tag has these attributes:

Attribute:	Description:
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Comment`	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MessageType`	Enter Block Transfer Read, Block Transfer Write, CIP Data Table Read, CIP Data Table Write, CIP Generic, PLC2 Unprotected Read, PLC2 Unprotected Write, PLC3 Typed Read, PLC3 Typed Write, PLC3 Word Range Read, PLC3 Word Range Write, PLC5 Typed Read, PLC5 Typed Write, PLC5 Word Range Read, PLC5 Word Range Write, SLC Typed Read, or SLC Typed Write. Specify: `MessageType := text`
`RequestedLength`	Specify the number of elements in the message instruction (0-32,767). Specify: `RequestedLength := value`

Attribute:	Description:
ConnectionPath	Specify the connection path to the other device. Specify: `ConnectionPath := string`
DF1DHFlag	If the communication method uses DH+, enter 1. If the communication method does not use DH+, enter 0. Specify: `DF1DHFlag := value`
LocalTag	Specify the tag name of the element in the local device. Specify: `LocalTag := text`
RemoteElement	Specify the tag name of the element in the remote device. Specify: `RemoteElement := value`
DHPlusSourceLink	If the communication method uses DH+, specify the source link (0-65,535). Specify: `DHPlusSourceLink := value`
DHPlusDestinationLink	If the communication method uses DH+, specify the destination link (0-65,535). Specify: `DHPlusDestinationLink := value`
DHPlusDestinationNode	If the communication method uses DH+, specify the destination node number (0-63 octal). Specify: `DHPlusDestinationNode := value`
DHPlusChannel	If the communication method uses DH+, specify the DH+ channel. Enter either A or B. Specify: `DHPlusChannel := letter`
CacheConnections	If the message is to cache connections, enter TRUE. If the message is not to cache connections, enter FALSE. Specify: `CacheConnections := text`
ServiceCode	If the message type is CIP Generic, specify the service code (0-255 hexadecimal). Specify: `ServiceCode := #16value`
ObjectType	If the message type is CIP Generic, specify the object type (0-65,535 hexadecimal). Specify: `ObjectType := 16#value`
TargetObject	If the message type is CIP Generic, specify the target object (0-65,535 decimal). Specify: `TargetObject := value`
AttributeNumber	If the message type is CIP Generic, specify the attribute number (0-65,535 hexadecimal). Specify: `AttributeNumber := 16#value`
DestinationTag	Specify the tag name of the destination element. Specify: `DestinationTag := text`

Specifying attributes for an AXIS_CONSUMED tag

The axis tags differ from the basic tag. The AXIS_CONSUMED tag has these attributes:

Attribute:	**Description:**
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Comment`	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MotionGroup`	Enter the name of the associated motion group, or enter <NA>. Specify: `MotionGroup := text`
`ConversionConstant`	Specify the number of feedback counts per position unit. Enter a real number from 1.0 to $1.0e^9$. Specify: `ConversionConstant := value`
`OutputCamExecutionTargets`	Specify the number of output cam execution targets (any positive number). Specify: `OutputCamExecutionTargets := text`
`AxisState`	Enter Axis-Ready, Direct Drive Control, Servo Control, Axis Faulted, or Axis Shutdown. Specify: `AxisState := text`
`PositionUnits`	Specify user-defined engineering units (rather than feedback units). Specify: `PositionUnits := text`
`AverageVelocityTimebase`	Specify the time in seconds for calculating the average velocity of the axis (any positive number). Specify: `AverageVelocityTimebase := value`
`RotaryAxis`	Specify the positioning mode for an axis. For this axis type, enter Linear. Specify: `RotaryAxis := text`
`PositionUnwind`	For a rotary axis, specify the distance (in feedback counts) used to perform electronic unwind (any positive number). For this axis type, enter 8000. Specify: `PositionUnwind := value`

Specifying attributes for an AXIS_VIRTUAL tag

The axis tags differ from the basic tag. The AXIS_VIRTUAL tag has these attributes:

Attribute:	**Description:**
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Comment`	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MotionGroup`	Enter the name of the associated motion group, or enter <NA>. Specify: `MotionGroup := text`
`ConversionConstant`	Specify the number of feedback counts per position unit. Enter a real number from 1.0 to $1.0e^9$. Specify: `ConversionConstant := value`
`OutputCamExecutionTargets`	Specify the number of output cam execution targets (any positive number). Specify: `OutputCamExecutionTargets := text`
`AxisState`	Enter Axis-Ready, Direct Drive Control, Servo Control, Axis Faulted, or Axis Shutdown. Specify: `AxisState := text`
`PositionUnits`	Specify user-defined engineering units (rather than feedback units). Specify: `PositionUnits := text`
`AverageVelocityTimebase`	Specify the time in seconds for calculating the average velocity of the axis (any positive number). Specify: `AverageVelocityTimebase := value`
`RotaryAxis`	Specify the positioning mode for an axis. Enter Linear or Rotary. Specify: `RotaryAxis := text`
`PositionUnwind`	For a rotary axis, specify the distance (in feedback counts) used to perform electronic unwind (any positive number). Specify: `PositionUnwind := value`
`HomeMode`	Specify the homing mode. Enter Passive, Active, or Absolute. Specify: `HomeMode := text`
`HomeDirection`	For active homing sequences, except for the immediate sequence type, specify the desired homing direction. Enter Uni-directional Forward, Bi-directional Forward, Uni-directional Reverse, or Bi-directional Reverse. Specify: `HomeDirection := text`
`HomeSequence`	Specify the event that will cause the home position to be set. Enter Immediate, Switch, Marker, or Switch-Marker. Specify: `HomeSequence := text`
`HomeConfigurationBits`	Specify the home configuration bits. Enter a hexadecimal number. Specify: `HomeConfigurationBits := 16#value`
`HomePosition`	Specify the desired absolute position, in positioning units, for the axis after the homing sequence is complete (any positive number). Specify: `HomePosition := value`

Attribute:	Description:
`HomeOffset`	Specify the desired offset (any positive number) in position units the axis is to move, upon completion of the homing sequence, to reach the home position. In most cases, this value will be zero. Specify: `HomeOffset := value`
`MaximumSpeed`	Specify the maximum speed (any positive number). Specify: `MaximumSpeed := value`
`MaximumAcceleration`	Specify the maximum acceleration rate of the axis in position units/second (any positive number). Specify: `MaximumAcceleration := value`
`MaximumDeceleration`	Specify the maximum deceleration rate of the axis in position units/second (any positive number). Specify: `MaximumDeceleration := value`
`ProgrammedStopMode`	Specify how a specific axis will stop when the controller changes mode or a motion group stop (MGS) instruction is executed. Enter Fast Disable, Fast Stop, Fast Shutdown, Hard Disable, or Hard Shutdown. Specify: `ProgrammedStopMode := text`

Specifying attributes for an AXIS_SERVO tag

The axis tags differ from the basic tag. The AXIS_SERVO tag has these attributes:

Attribute:	Description:
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Comment`	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MotionGroup`	Enter the name of the associated motion group, or enter <NA>. Specify: `MotionGroup := text`
`MotionModule`	Enter the name of the associated motion module, or enter <NA>. Specify: `MotionModule := text`
`ConversionConstant`	Specify the number of feedback counts per position unit. Enter a real number from 1.0 to $1.0e^9$. Specify: `ConversionConstant := value`
`OutputCamExecutionTargets`	Specify the number of output cam execution targets (any positive number). Specify: `OutputCamExecutionTargets := text`
`AxisState`	Enter Axis-Ready, Direct Drive Control, Servo Control, Axis Faulted, or Axis Shutdown. Specify: `AxisState := text`
`PositionUnits`	Specify user-defined engineering units (rather than feedback units). Specify: `PositionUnits := text`
`AverageVelocityTimebase`	Specify the time in seconds for calculating the average velocity of the axis (any positive number). Specify: `AverageVelocityTimebase := value`

Attribute:	Description:
RotaryAxis	Specify the positioning mode for an axis. Enter Rotary or Linear. Specify: `RotaryAxis := text`
PositionUnwind	For a rotary axis, specify the distance (in feedback counts) used to perform electronic unwind (any positive number). Specify: `PositionUnwind := value`
HomeMode	Specify the homing mode. Enter Passive, Active, or Absolute. Specify: `HomeMode := text`
HomeDirection	For active homing sequences, except for the immediate sequence type, specify the desired homing direction. Enter Uni-directional Forward, Bi-directional Forward, Uni-directional Reverse, or Bi-directional Reverse. Specify: `HomeDirection := text`
HomeSequence	Specify the event that will cause the home position to be set. Enter Immediate, Switch, Marker, or Switch-Marker. Specify: `HomeSequence := text`
HomeConfigurationBits	Specify the home configuration bits. Enter a hexadecimal number. Specify: `HomeConfigurationBits := 16#value`
HomePosition	Specify the desired absolute position, in positioning units, for the axis after the homing sequence is complete (any positive number). Specify: `HomePosition := value`
HomeOffset	Specify the desired offset (any positive number) in position units the axis is to move, upon completion of the homing sequence, to reach the home position. In most cases, this value will be zero. Specify: `HomeOffset := value`
HomeSpeed	Specify the speed of the jog profile used in the first leg of the homing sequence (any positive number). The homing speed should be less than the maximum speed and greater than zero. Specify: `HomeSpeed := value`
HomeReturnSpeed	Specify speed of the jog profile used in the return leg(s) of an active homing sequence (any positive number). The return speed should be less than the maximum speed and greater than zero. Specify: `HomeReturnSpeed := value`
MaximumSpeed	Specify the maximum speed (any positive number). Specify: `MaximumSpeed := value`
MaximumAcceleration	Specify the maximum acceleration rate of the axis in position units/second (any positive number). Specify: `MaximumAcceleration := value`
MaximumDeceleration	Specify the maximum deceleration rate of the axis in position units/second (any positive number). Specify: `MaximumDeceleration := value`
ProgrammedStopMode	Specify how a specific axis will stop when the controller changes mode or a motion group stop (MGS) instruction is executed. Enter Fast Disable, Fast Stop, Fast Shutdown, Hard Disable, or Hard Shutdown. Specify: `ProgrammedStopMode := text`
AxisType	Specify the intended use of the axis. Enter Servo or Feedback Only. Specify: `AxisType := text`
ServoLoopConfiguration	Specify the configuration of the loop. Enter Position Servo. Specify: `ServoLoopConfiguration := text`

Attribute:	Description:
`ExternalDriveType`	Specify the drive type for the servo loop. Enter Torque or Velocity. Specify: `ExternalDriveType := text`
`FaultConfigurationBits`	Specify the fault configuration bits. Enter a hexadecimal number. Specify: `FaultConfigurationBits := 16#value`
`AxisInfoSelect1`	Specify an axis attribute to transmit, along with the actual position data, to the controller. Enter <none>, Position Command, Position Feedback, Aux Position Feedback, Position Error, Position Int. Error, Velocity Command, Velocity Feedback, Velocity Error, Velocity Int. Error, Accel. Command, Accel. Feedback, Servo Output Level, Marker Distance, Torque Command, Torque Feedback, Positive Dynamic Torque Limit, Negative Dynamic Torque Limit, Motor Capacity, Drive Capacity, Power Capacity, Bus Regulator Capacity, Motor Electrical Angle, Torque Limit Source, DC Bus Voltage. Specify: `AxisInfoSelect1 := text`
`AxisInfoSelect2`	Specify a second axis attribute to transmit, along with the actual position data, to the controller. Enter <none>, Position Command, Position Feedback, Aux Position Feedback, Position Error, Position Int. Error, Velocity Command, Velocity Feedback, Velocity Error, Velocity Int. Error, Accel. Command, Accel. Feedback, Servo Output Level, Marker Distance, Torque Command, Torque Feedback, Positive Dynamic Torque Limit, Negative Dynamic Torque Limit, Motor Capacity, Drive Capacity, Power Capacity, Bus Regulator Capacity, Motor Electrical Angle, Torque Limit Source, DC Bus Voltage. Specify: `AxisInfoSelect2 := text`
`ServoPolarityBits`	Specify the servo polarity bits. Enter a hexadecimal number. Specify: `ServoPolarityBits := 16#value`
`VelocityFeedforwardGain`	Specify the velocity feedforward gain (any positive number). Specify: `VelocityFeedforwardGain := value`
`AccelerationFeedforwardGain`	Specify the acceleration feedforward gain (any positive number). Specify: `AccelerationFeedforwardGain := value`
`PositionProportionalGain`	Specify the position proportional gain (any positive number). Specify: `ProportionalPositionGain := value`
`PositionIntegralGain`	Specify the position integral gain (any positive number). Specify: `PositionIntegralGain := value`
`VelocityProportionalGain`	Specify the velocity proportional gain (any positive number). Specify: `VelocityProportionalGain := value`
`VelocityIntegralGain`	Specify the velocity integral gain (any positive number). Specify: `VelocityIntegralGain := value`
`VelocityScaling`	Specify the velocity scaling attribute that is used to convert the output of the servo loop into equivalent voltage to an external velocity servo drive. Specify: `VelocityScaling := value`
`TorqueScaling`	Specify the torque scaling attribute that is used to convert the acceleration of the servo loop into equivalent % rated torque to the motor. Specify: `TorqueScaling := value`
`OutputLPFilterBandwidth`	Specify the bandwidth in Hertz of the servo's low-pass digital output filter. Specify: `OutputLPFilterBandwidth := value`
`IntegratorHoldEnable`	Enter Disabled or Enabled. Specify: `IntegratorHoldEnable := value`
`MaximumPositiveTravel`	Specify the maximum positive position (any positive number) to be used for software overtravel checking, in position units. Specify: `MaximumPositiveTravel := value`

Attribute:	Description:
MaximumNegativeTravel	Specify the maximum negative position (any positive number) to be used for software overtravel checking, in position units. Specify: `MaximumNegativeTravel := value`
PositionErrorTolerance	Specify the how position error the servo module will tolerate (any positive number) before issuing a position error fault. Specify: `PositionErrorTolerance := value`
PositionLockTolerance	Specify the maximum position error the servo module will accept (any positive number) in order to indicate that the position lock status bit is set. Specify: `PositionLockTolerance := value`
OutputLimit	Specify the maximum servo output voltage of a physical axis (any positive number). Specify: `OutputLimit := value`
OutputOffset	Specify a fixed voltage value (any positive number not to exceed ±10V) to add to the servo output value to correct axis drift. Specify: `OutputOffset := value`
VelocityOffset	Specify a dynamic velocity correction to the output of the position servo loop, in position units/second (any positive number). Specify: `VelocityOffset := value`
TorqueOffset	Specify a dynamic torque command correction to the output of the velocity servo loop as a percentage of the velocity servo loop output (any positive number). Specify: `TorqueOffset := value`
FrictionCompensation	Specify the percentage (any positive number) of output level added to a positive current servo output value, or subtracted from a negative current servo output value, for the purpose of moving an axis that is stuck in place due to static friction. Specify: `FrictionCompensation := value`
HardOvertravelFaultAction	Specify the fault action taken when a hardware overtravel error occurs. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `HardOvertravelFaultAction := text`
SoftOvertravelFaultAction	Specify the fault action taken when a software overtravel error occurs. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `SoftOvertravelFaultAction := text`
PositionErrorFaultAction	Specify the fault action taken when the position error exceeds the position tolerance set for the axis. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `PositionErrorFaultAction := text`
FeedbackFaultAction	Specify the fault action to be taken when a feedback loss condition is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `FeedbackFaultAction := text`
FeedbackNoiseFaultAction	Specify the fault action to be taken when excessive feedback noise is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `FeedbackNoiseFaultAction := text`
DriveFaultAction	Specify the fault action to be taken when a drive fault condition is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `DriveFaultAction := text`
TestIncrement	Specify the amount of distance traversed by the axis when executing the output and feedback test (any positive number). Specify: `TestIncrement := value`
TuningTravelLimit	Specify the tuning travel limit in revolutions (any positive number). Specify: `TuningTravelLimit := value`

Attribute:	Description:
TuningSpeed	Specify the tuning speed in revolutions per second (any positive number). Specify: `TuningSpeed := value`
TuningTorque	Specify the tuning torque % rated (any positive number). Specify: `TuningSpeed := value`
DampingFactor	Specify the damping factor (any positive number). Specify: `DampingFactor := value`
DriveModelTimeConstant	Specify the drive model time constant (any positive number). Specify: `DriveModelTimeConstant := value`
PositionServoBandwidth	Specify the maximum allowable value for position bandwidth (any positive number), given the damping factor. This parameter is disabled if the loop configuration is set to velocity. Specify: `PositionServoBandwidth := value`
VelocityServoBandwidth	Specify the unity gain bandwidth that is to be used to calculate the subsequent gains for a motion apply axis tuning (MAAT) instruction (any positive number). Specify: `VelocityServoBandwidth := value`
TuningConfigurationBits	Specify the tuning configuration bits. Enter a hexadecimal number. Specify: `TuningConfigurationBits := 16#value`

Specifying attributes for an AXIS_SERVO_DRIVE tag

The axis tags differ from the basic tag. The AXIS_SERVO_DRIVE tag has these attributes

Attribute:	Description:
Description	Provide information about the tag. Specify: `Description := "text"`
Comment	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
MotionGroup	Enter the name of the associated motion group, or enter <NA>. Specify: `MotionGroup := text`
MotionModule	Enter the name of the associated motion module, or enter <NA>. Specify: `MotionModule := text`
RotationalPosResolution	Specify the number of counts per motor revolution (1 to [2^{32}-1]). Specify: `RotationalPosResolution := text`
ConversionConstant	Specify the number of feedback counts per position unit. Enter a real number from 1.0 to 1.0e^{9}. Specify: `ConversionConstant := value`
AmplifierCatalogNumber	Specify the catalog number of the amplifier to which this axis is connected. Specify: `AmplifierCatalogNumber := text`
MotorCatalogNumber	Specify the catalog number of the motor to which this axis is connected or enter <NONE>. Specify: `MotorCatalogNumber := text`

Attribute:	Description:
OutputCamExecutionTargets	Specify the number of output cam execution targets (any positive number). Specify: `OutputCamExecutionTargets := text`
AxisState	Enter Axis-Ready, Direct Drive Control, Servo Control, Axis Faulted, or Axis Shutdown. Specify: `AxisState := text`
PositionUnits	Specify user-defined engineering units (rather than feedback units). Specify: `PositionUnits := text`
AverageVelocityTimebase	Specify the time in seconds for calculating the average velocity of the axis (any positive number). Specify: `AverageVelocityTimebase := value`
RotaryAxis	Specify the positioning mode for an axis. Enter Rotary or Linear. Specify: `RotaryAxis := text`
PositionUnwind	For a rotary axis, specify the distance (in feedback counts) used to perform electronic unwind (any positive number). Specify: `PositionUnwind := value`
HomeMode	Specify the homing mode. Enter Passive, Active, or Absolute. Specify: `HomeMode := text`
HomeDirection	For active homing sequences, except for the immediate sequence type, specify the desired homing direction. Enter Uni-directional Forward, Bi-directional Forward, Uni-directional Reverse, or Bi-directional Reverse. Specify: `HomeDirection := text`
HomeSequence	Specify the event that will cause the home position to be set. Enter Immediate, Switch, Marker, or Switch-Marker. Specify: `HomeSequence := text`
HomeConfigurationBits	Specify the home configuration bits. Enter a hexadecimal number. Specify: `HomeConfigurationBits := 16#value`
HomePosition	Specify the desired absolute position, in positioning units, for the axis after the homing sequence is complete (any positive number). Specify: `HomePosition := value`
HomeOffset	Specify the desired offset (any positive number) in position units the axis is to move, upon completion of the homing sequence, to reach the home position. In most cases, this value will be zero. Specify: `HomeOffset := value`
HomeSpeed	Specify the speed of the jog profile used in the first leg of the homing sequence (any positive number). The homing speed should be less than the maximum speed and greater than zero. Specify: `HomeSpeed := value`
HomeReturnSpeed	Specify speed of the jog profile used in the return leg(s) of an active homing sequence (any positive number). The return speed should be less than the maximum speed and greater than zero. Specify: `HomeReturnSpeed := value`
MaximumSpeed	Specify the maximum speed (any positive number). Specify: `MaximumSpeed := value`
MaximumAcceleration	Specify the maximum acceleration rate of the axis in position units/second (any positive number). Specify: `MaximumAcceleration := value`
MaximumDeceleration	Specify the maximum deceleration rate of the axis in position units/second (any positive number). Specify: `MaximumDeceleration := value`

Attribute:	Description:
`ProgrammedStopMode`	Specify how a specific axis will stop when the controller changes mode or a motion group stop (MGS) instruction is executed. Enter Fast Disable, Fast Stop, Fast Shutdown, Hard Disable, or Hard Shutdown. Specify: `ProgrammedStopMode := text`
`AxisType`	Specify the intended use of the axis. Enter Servo or Feedback Only. Specify: `AxisType := text`
`ServoLoopConfiguration`	Specify the configuration of the loop. Enter Custom, Position Servo, Aux Position Servo, Dual Position Servo, Aux Command Servo, Dual Command Servo, Velocity Servo, or Torque Servo. Specify: `ServoLoopConfiguration := text`
`FaultConfigurationBits`	Specify the fault configuration bits. Enter a hexadecimal number. Specify: `FaultConfigurationBits := 16#value`
`AxisInfoSelect1`	Specify an axis attribute to transmit, along with the actual position data, to the controller. Enter <none>, Position Command, Position Feedback, Aux Position Feedback, Position Error, Position Int. Error, Velocity Command, Velocity Feedback, Velocity Error, Velocity Int. Error, Accel. Command, Accel. Feedback, Servo Output Level, Marker Distance, Torque Command, Torque Feedback, Positive Dynamic Torque Limit, Negative Dynamic Torque Limit, Motor Capacity, Drive Capacity, Power Capacity, Bus Regulator Capacity, Motor Electrical Angle, Torque Limit Source, DC Bus Voltage. Specify: `AxisInfoSelect1 := text`
`AxisInfoSelect2`	Specify a second axis attribute to transmit, along with the actual position data, to the controller. Enter <none>, Position Command, Position Feedback, Aux Position Feedback, Position Error, Position Int. Error, Velocity Command, Velocity Feedback, Velocity Error, Velocity Int. Error, Accel. Command, Accel. Feedback, Servo Output Level, Marker Distance, Torque Command, Torque Feedback, Positive Dynamic Torque Limit, Negative Dynamic Torque Limit, Motor Capacity, Drive Capacity, Power Capacity, Bus Regulator Capacity, Motor Electrical Angle, Torque Limit Source, DC Bus Voltage. Specify: `AxisInfoSelect2 := text`
`VelocityFeedforwardGain`	Specify the velocity feedforward gain (any positive number). Specify: `VelocityFeedforwardGain := value`
`AccelerationFeedforwardGain`	Specify the acceleration feedforward gain (any positive number). Specify: `AccelerationFeedforwardGain := value`
`PositionProportionalGain`	Specify the position proportional gain (any positive number). Specify: `ProportionalPositionGain := value`
`PositionIntegralGain`	Specify the position integral gain (any positive number). Specify: `PositionIntegralGain := value`
`VelocityProportionalGain`	Specify the velocity proportional gain (any positive number). Specify: `VelocityProportionalGain := value`
`VelocityIntegralGain`	Specify the velocity integral gain (any positive number). Specify: `VelocityIntegralGain := value`
`TorqueScaling`	Specify the torque scaling attribute that is used to convert the acceleration of the servo loop into equivalent % rated torque to the motor. Specify: `TorqueScaling := value`
`OutputLPFilterBandwidth`	Specify the bandwidth in Hertz of the servo's low-pass digital output filter. Specify: `OutputLPFilterBandwidth := value`
`IntegratorHoldEnable`	Enter Disabled or Enabled. Specify: `IntegratorHoldEnable := value`

Attribute:	Description:
MaximumPositiveTravel	Specify the maximum positive position (any positive number) to be used for software overtravel checking, in position units. Specify: `MaximumPositiveTravel := value`
MaximumNegativeTravel	Specify the maximum negative position (any positive number) to be used for software overtravel checking, in position units. Specify: `MaximumNegativeTravel := value`
PositionErrorTolerance	Specify the how position error the servo module will tolerate (any positive number) before issuing a position error fault. Specify: `PositionErrorTolerance := value`
PositionLockTolerance	Specify the maximum position error the servo module will accept (any positive number) in order to indicate that the position lock status bit is set. Specify: `PositionLockTolerance := value`
VelocityOffset	Specify a dynamic velocity correction to the output of the position servo loop, in position units/second (any positive number). Specify: `VelocityOffset := value`
TorqueOffset	Specify a dynamic torque command correction to the output of the velocity servo loop as a percentage of the velocity servo loop output (any positive number). Specify: `TorqueOffset := value`
FrictionCompensation	Specify the percentage (any positive number) of output level added to a positive current servo output value, or subtracted from a negative current servo output value, for the purpose of moving an axis that is stuck in place due to static friction. Specify: `FrictionCompensation := value`
HardOvertravelFaultAction	Specify the fault action taken when a hardware overtravel error occurs. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `HardOvertravelFaultAction := text`
SoftOvertravelFaultAction	Specify the fault action taken when a software overtravel error occurs. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `SoftOvertravelFaultAction := text`
PositionErrorFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `PositionErrorFaultAction := text`
FeedbackFaultAction	Specify the fault action to be taken when a feedback loss condition is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `FeedbackFaultAction := text`
FeedbackNoiseFaultAction	Specify the fault action to be taken when excessive feedback noise is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: `FeedbackNoiseFaultAction := text`
TestIncrement	Specify the amount of distance traversed by the axis when executing the output and feedback test (any positive number). Specify: `TestIncrement := value`
TuningTravelLimit	Specify the tuning travel limit in revolutions (any positive number). Specify: `TuningTravelLimit := value`
TuningSpeed	Specify the tuning speed in revolutions per second (any positive number). Specify: `TuningSpeed := value`
TuningTorque	Specify the tuning torque % rated (any positive number). Specify: `TuningTorque := value`
DampingFactor	Specify the damping factor (any positive number). Specify: `DampingFactor := value`

Attribute:	Description:
DriveModelTimeConstant	Specify the drive model time constant (any positive number). Specify: `DriveModelTimeConstant := value`
PositionServoBandwidth	Specify the maximum allowable value for position bandwidth (any positive number), given the damping factor. This parameter is disabled if the loop configuration is set to velocity. Specify: `PositionServoBandwidth := value`
VelocityServoBandwidth	Specify the unity gain bandwidth that is to be used to calculate the subsequent gains for a motion apply axis tuning (MAAT) instruction (any positive number). Specify: `VelocityServoBandwidth := value`
TuningConfigurationBits	Specify the tuning configuration bits. Enter a hexadecimal number. Specify: `TuningConfigurationBits := 16#value`
TorqueLimitSource	Enter Not Limited, Negative Limit, Positive Limit, Bridge Limit, I(t) Limit, or Motor Limit. Specify: `TorqueLimitSource := text`
PositionDataScaling	Specify the scaling method used on position values (0-255). Specify: `PositionDataScaling := value`
PositionDataScalingFactor	Specify the scaling factor for all position data in a drive (1-65535). Specify: `PositionDataScalingFactor := value`
PositionDataScalingExp	Specify the scaling exponent for all position data in a drive (-32768 to 32767). Specify: `PositionDataScalingExp := value`
VelocityDataScaling	Specify the scaling method to use for all velocity values (0-127). Specify: `VelocityDataScaling := value`
VelocityDataScalingFactor	Specify the scaling factor for all velocity data (1-65535). Specify: `VelocityDataScalingFactor := value`
VelocityDataScalingExp	Specify the scaling exponent for all velocity data (-32768 to 32767). Specify: `VelocityDataScalingExp := value`
AccelerationDataScaling	Specify the scaling method for all acceleration values (0-127). Specify: `AccelerationDataScaling := value`
AccelerationData ScalingFactor	Specify the scaling factor for all acceleration data (1-65535). Specify: `AccelerationDataScalingFactor := value`
AccelerationDataScalingExp	Specify the scaling exponent for all acceleration data (-32768 to 32767). Specify: `AccelerationDataScalingExp := value`
TorqueDataScaling	Specify the scaling method for all torque values (0-127). Specify: `TorqueDataScaling := value`
TorqueDataScalingFactor	Specify the scaling factor for all torque values (1-65535). Specify: `TorqueDataScalingFactor := value`
TorqueDataScalingExp	Specify the scaling exponent for all torque values (-32768 to 32767). Specify: `TorqueDataScalingExp := value`
DrivePolarity	Specify the polarity of the servo loop of the drive. Enter Custom, Positive, or Negative. Specify: `DrivePolarity := text`
MotorFeedbackType	Specify the type of motor associated with the selected motor (MotorCatalogNumber). If you specify <NONE> for the motor, you must specify a feedback type. Specify: `MotorFeedbackType := value`
MotorFeedbackResolution	Specify the resolution of the motor (1-2147483647). Specify: `MotorFeedbackResolution := value`
AuxFeedbackType	Specify the type of auxiliary feedback device. Specify: `AuxFeedbackType := value`

Attribute:	Description:
AuxFeedbackResolution	Specify the resolution of the auxiliary feedback device (1-2147483647). Specify: AuxFeedbackResolution := *value*
OutputNotchFilterFrequency	Specify the frequency of the drive's digital notch filer (0 -1000.0). Specify: OutputNotchFilterFrequency := *value*
VelocityDroop	Specify the velocity droop (any positive number). Specify: VelocityDroop := *value*
VelocityLimitBipolar	Specify the velocity limit symmetrically in both directions (any positive number). Specify: VelocityLimitBipolar := *value*
AccelerationLimitBipolar	Specify the acceleration and deceleration limits for the drive (any positive number). Specify: AccelerationLimitBipolar := *value*
TorqueLimitBipolar	Specify the torque limit symmetrically in both directions (0 - 1000.0). Specify: TorqueLimitBipolar := *value*
VelocityLimitPositive	Specify the maximum allowable velocity in the positive direction (any positive number). Specify: VelocityLimitPositive := *value*
VelocityLimitNegative	Specify the maximum allowable velocity in the negative direction (any positive number). Specify: VelocityLimitNegative := *value*
VelocityThreshold	Specify the velocity threshold limit (any positive number). Specify: VelocityThreshold := *value*
VelocityWindow	Specify the limits of the velocity window (any positive number). Specify: VelocityWindow := *value*
VelocityStandstillWindow	Specify the velocity limit for the standstill window (any positive number). Specify: VelocityStandstillWindow := *value*
AccelerationLimitPositive	Specify the maximum acceleration ability of the drive (any positive number). Specify: AccelerationLimitPositive := *value*
AccelerationLimitNegative	Specify the maximum acceleration ability of the drive (any negative number). Specify: AccelerationLimitNegative := *value*
TorqueLimitPositive	Specify the maximum torque in the positive direction (0-1000.0). Specify: TorqueLimitPositive := *value*
TorqueLimitNegative	Specify the maximum torque in the negative direction (-1000.0 - 0). Specify: TorqueLimitNegative := *value*
TorqueThreshold	Specify the torque threshold (0-1000.0). Specify: TorqueThreshold := *value*
DriveThermalFaultAction	Specify the fault action to be taken when a drive thermal fault is detected, for an axis configured as servo. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: DriveThermalFaultAction := *value*
MotorThermalFaultAction	Specify the fault action to be taken when a motor thermal fault is detected, for an axis configured as servo. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: MotorThermalFaultAction := *value*
StoppingTorque	Specify the amount of torque available to stop the motor (0-1000). Specify: StoppingTorque := *value*
StoppingTimeLimit	Specify the maximum amount of time that the drive amplifier will remain enabled while trying to stop (0-6553.5). Specify: StoppingTimeLimit := *value*

Attribute:	Description:
BrakeEngageDelayTime	Specify the amount of time that the drive maintains torque when the servo axis is disabled and the drive decelerates to a minimum speed (0-6.5535). Specify: `BrakeEngageDelayTime := value`
BrakeReleaseDelayTime	Specify amount of time that the drive ignores command values from the controller when the servo axis is enabled and the drive activates the torque (0-6.5535). Specify: `BrakeReleaseDelayTime := value`
PowerSupplyID	Specify the power supply ID (any positive number). Specify: `PowerSupplyID := value`
BusRegulatorID	Specify the bus regulator ID (any positive number). Specify: `BusRegulatorID := value`

Defining TAG initial values

The `initial_value` format follows the C-language initialization syntax, except that you use square brackets instead of curly brackets. The following table shows some examples of entering initial values.

If the tag is:	Enter:
single, atomic value	[*Value*]
structure with three members	[*Value1*, *Value2*, *Value3*]
structure with a nested structure	[*Value1*, [*Value2*, *Value3*], *Value4*]
structure with a nested array	[*Value1*, [*ArrayValue1*, *ArrayValue2*], *Value3*]

The initial value for a string value identifies the number of characters in the string and the text string. The format for a string TAG is:

`<tag_name> : STRING := [<number>, 'string_text$00 ... $00'];`

Where:

Item:	Identifies:
tag_name	name of the string tag
STRING	the STRING data type
number	number of characters in the string
string_text	text of the string
$00	the string is padded with $00 to fill its maximum of 82 characters Each $00 equals one character not used in the string. The entire text string, including the $00 characters, is enclosed in single quotation marks.

For example:

```
TAG
      sourcea_string : STRING  :=
[5,'hello$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00
$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00
$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00
$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00'
      ];
      sourceb_string : STRING  := [11,'how are
you$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00
$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00
$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00$00'
      ];
END_TAG
```

Defining a comment for a TAG component

The comment attribute of a tag declaration lets you provide information about a component of the tag, such as a specific bit, array element, or structure member. For example:

To add a comment to this operand:	Enter:
bit 3 of a tag	COMMENT.3 := "*description*"
element 8 of an array tag	COMMENT[8] := "*description*"
preset value of a tag	COMMENT.PRE := "*description*"

TAG guidelines

Keep these guidelines in mind when defining a tag:

- Tags must be defined after devices (if there are no devices, then after the data types) within the controller body.
- Base tags and aliases can be defined out of order within a tag block.
- You cannot define a 2nd dimension without a 1st dimension or a 3rd dimension without a 2nd dimension.
- The initial values must comply with the tag type and dimensions.
- Whitespace can not occur within the initial values or within the type/dimension specifier.

TAG examples

```
TAG
       bits : MySint := [0];
       dest : INT (RADIX := Decimal) := 0;
       overflow OF bits.MyBit0 (RADIX := Binary);
       source : REAL (RADIX := Exponential) := 0.0;
       timer : TIMER[3] := [[0,0,100],[0,10,100],[0,0,50]];
END_TAG
```

This example shows forced tag data:

```
TAG
       dint_a : DINT (RADIX := Decimal) := 0;
       int_a : INT (RADIX := Decimal) := 0;
       tag_a : UDT_A (ProduceCount := 2) := [0,0],
       TagForceData := [0,0,0,0,1,0,-1,-1,1,0,-72,34];
END_TAG
```

Defining a Program

A PROGRAM component follows this structure:

```
PROGRAM <program_name> [Attributes]
       [TAG declaration]
       [ROUTINE declaration]
       [FBD_ROUTINE declaration]
END_PROGRAM
```

Where:

Item:	Identifies:
`program_name`	the program
`Attributes`	attributes of the program (such as MAIN or FAULT) can also contain a description of the program separate each attribute with a comma (,)
`TAG`	program-scoped tags follows same format as controller-scoped tags see page 3-10
`ROUTINE`	ladder logic routines for this program see page 3-33
`FBD_ROUTINE`	function block diagram routines for this program see page 3-33

You can intermix ROUTINE and FBD_ROUTINE declarations. The FBD_ROUTINE declarations do not have to follow the ROUTINE declarations.

Specifying PROGRAM attributes

You can specify these attributes for a PROGRAM:

Attribute:	Description:
`Description`	Provide information about the program. Specify: `Description := "text"`
`Main`	Name of the main routine of the program. Specify: `Main := name`
`Fault`	Name of the program fault routine, if any. Specify: `Fault := name`

PROGRAM guidelines

Keep in mind these guidelines when defining a program:

- The MAIN and FAULT attributes can be defined in any order.
- The TAG declaration block must occur before the routine block.
- All tag collection declaration blocks that occur in a program definition block are imported as tags of a given program and can only be seen by routines under that program. Controller tags, on the other hand can be seen by routines in any program.

PROGRAM example

```
PROGRAM Prg1 (Main := RoutineB, Description := "I $'am$'" "
$0034 a $"program$"")
       TAG
       st11 : DINT (RADIX := Decimal, ProduceCount := 0) := 2;
       st12 : BOOL (RADIX := Binary, ProduceCount := 0) :=
2#00000000;
       END_TAG

       ROUTINE RoutineA
       JSR(_2_LADDER, 0);
       END_ROUTINE

       ROUTINE RoutineB
              RC: "$L ** ;MORE $";STUFF" do not include "more";
              xic(st11) ote(st12);
       END_ROUTINE
END_PROGRAM
```

Defining a Ladder Logic Routine

A ladder logic ROUTINE component follows this structure:

```
ROUTINE <routine_name> [Attributes]
       < logic >
END_ROUTINE
```

Where:

Item:	Identifies:
routine_name	the routine
Attributes	attributes of the routine can also contain a description of the routine separate each attribute with a comma (,)
logic	ladder logic

For the syntax of entering ladder logic see chapter 4.

Specifying ROUTINE attributes

You can specify these attributes for a ROUTINE:

Attribute:	Description:
Description	Provide information about the routine. Specify: `Description := "text"`

Defining a Function Block Diagram Routine

A function block FBD_ROUTINE component follows this structure:

```
FBD_ROUTINE <routine_name> [Attributes]
       <logic>
END_FBD_ROUTINE
```

Where:

Item:	Identifies:
routine_name	the routine
Attributes	attributes of the function block routine (such as sheet size or sheet orientation) can also contain a description of the routine separate each attribute with a comma (,)
logic	function block logic

For the syntax of entering function block logic, see chapter 5.

Specifying FBD_ROUTINE attributes

You can specify these attributes for a FBD_ROUTINE:

Attribute:	Description:
`Description`	Provide information about the routine. Specify: `Description := "text"`
`SheetSize`	Select one of these sizes: • Letter (8.5x11in) • Legal (8.5x14in) • Tabloid (11x17.in) • A4 (210x297mm) • A3 (297x420mm) Specify: `SheetSize := size`
`SheetOrientation`	Select the orientation of the sheet as Portrait or Landscape. Specify: `SheetOrientation := type`

Defining a Task

The maximum number of tasks depends on the of controller:

Controller:	Maximum Number of Tasks:
ControlLogix	32
SoftLogix5800	32
FlexLogix	8
CompactLogix (L20, L30)	4
DriveLogix	4

A TASK component follows this structure:

```
TASK <task_name> [Attributes]
      <program_name>;
END_TASK
```

Where:

Item:	Identifies:
`task_name`	the task
`Attributes`	attributes of the task can also contain a description of the task separate each attribute with a comma (,)
`program_name`	each program within the task all program names are followed by a semi colon (;)

Specifying TASK attributes

You can specify these attributes for a TASK:

Attribute:	Description:
`Description`	Provide information about the task. Specify: `Description := "text"`
`Type`	Specify the type of task (PERIODIC or CONTINUOUS). There can be only one continuous task. Specify: `Type := type`
`Watchdog`	Enter the watchdog timeout for the task (1000-2,000,000,000 us). Specify: `Watchdog := number`
`Priority`	Specify the priority of a periodic task (1-15) Specify: `Priority := number`
`Rate`	If the task is a periodic task, specify how often to run the task (1000-2,000,000,000 us). Specify: `Rate := number`

TASK guidelines

Keep these guidelines in mind when defining a task:

- Tasks must be defined after programs and before controller objects.
- There can be at most 32 tasks.
- There can only be one continuous task.
- A program can be scheduled under only one task.
- Scheduled programs must be defined - i.e. must exist.

TASK example

```
TASK joe (TYPE := PERIODIC,  PRIORITY := 8, RATE := 10000)
       sue;
       betty;
END_TASK
```

The task attributes (Type, Priority, Rate, and Watchdog) can be defined in any order. The list of programs scheduled for a task are listed in the task declarations block, as shown above. The programs are executed in the order they are specified.

Defining a Controller Object

A CONFIG component defines controller objects and follows this structure:

```
CONFIG <object_name> [Attributes]
END_CONFIG
```

Where:

Item:	Identifies:
`object_name`	the controller object see the list of attributes below for a list of valid controller objects
`Attributes`	attributes of the controller object can also contain a description of the controller object separate each attribute with a comma (,)

Controller objects are optional. There can be only one of each controller object you choose to define. Controller objects appear at the end of the import/export file.

Specifying CONFIG attributes

The attributes depend on the type on CONFIG object you specify. Some objects do not have any attributes. The following table lists those objects that have attributes and descriptions of each:

Object:	Attribute:	Description:
ASCII	XONXOFFEnable	Specify whether to regulate the flow of incoming data. Enter 0 to disable XON/XOFF; enter 1 to enable XON/XOFF. Specify: `XONXOFFEnable := value`
	DeleteMode	Specify the delete mode. Enter 0 for Ignore; enter 1 for CRT; or enter 2 for Printer. Specify: `DeleteMode := value`
	EchoMode	Specify whether to echo data back to the device from which it was sent. Enter 0 to disable; enter 1 to enable. Specify: `EchoMode := value`
	TerminationChars	Specify the characters that designate the end of a line. Specify: `TerminationChars := value`
	AppendChars	Specify the characters to append to the end of a line. Specify: `AppendChars := value`
	NumberOfAppendChars	Specify the number of append characters. Enter 0, 1, or 2. Specify: `NumberOfAppendChars := value`
	BufferSize	Specify the maximum size of the data array (1-65535 bytes) that you plan to send and receive. Specify: `BufferSize := value`
ControllerDevice	none	none
CST	SystemTimeMasterID	Specify whether the controller is the coordinated system time master. Enter 16#0000 is the controller is not the CST master; enter 16#0001 if the controller is the CST master. Specify: `CST := 16#value`
DF1	DuplicateDetection	Specify whether to enable duplicate message detection, which ignores duplicate messages. Enter 0 to disable; enter 1 to enable. Specify: `DuplicateDetection := value`
	ErrorDetection	Specify the error detection method. Enter BCC Error or CRC Error. Specify: `ErrorDetection := text`
	EnbeddedResponseEnable	Specify the response method. Enter 0 to autodetect; enter 1 to enable. Specify: `EnbeddedResponseEnable := value`
	DF1Mode	Specify the DF1 mode. Enter Pt to Pt, Master, or Slave. Specify: `DF1Mode := value`
	ACKTimeout	Specify the time to wait for an acknowledgment to a message transmission. Enter an increment of 20ms (0-65535). Specify: `ACKTimeout := value`
	NAKReceiveLimit	Specify the number of NAKS (0-255) the controller can receive in response to a message before stopping transmission. Specify: `NAKReceiveLimit := value`

Object:	Attribute:	Description:
DF1 (continued)	ENQTransmit	Specify the number of inquiries (0-127) you want the controller to send after an ACK timeout. Specify: `ENQTransmit := value`
	TransmitRetries	Specify the number of attempted retires (0-255) without getting an acknowledgement before the message is deemed undeliverable. Specify: `TransmitRetries := value`
	StationAddress	Specify the current station link address (0-254). Specify: `StationAddress := value`
	ReplyMessageWait	Specify the time the master waits after receiving an acknowledgment to a master-initiated message before polling the slave for a response. Enter an increment of 20ms (0-65535). Specify: `ReplyMessageWait := value`
	PollingMode	Specify the polling mode. Enter on of these: • 1 for Message Based (slave can initiate messages) • 2 for Message Based (slave cannot initiate messages) • 3 for Standard (multiple message transfer for node scan) • 4 for Standard (single message transfer per node scan) Specify: `PollingMode := value`
	MasterMessageTransmit	Specify when the master transmits. Enter 0 to transmit between station polls; enter 1 to transmit in poll sequence. Specify: `MasterMessageTransmit := value`
	NormalPollNodeFile	Specify the tag name of the structure that contains the normal poll node list. Or enter <NA>. Specify: `NormalPollNodeFile := value`
	NormalPollGroupSize	Specify the total number (0-255) of active stations polled from the poll list. Specify: `NormalPollGroupSize := value`
	PriorityPollNodeFile	Specify the tag name of the structure that contains the priority poll node list. Or enter <NA>. Specify: `PriorityPollNodeFile := value`
	ActiveStationFile	Specify the tag name of the structure that contains the status (active or non-active) of each node. Or enter <NA>. Specify: `ActiveStationFile := value`
	SlavePollTimeout	Specify the amount of time the master waits for an acknowledgement to a message sent to a slave. Enter an increment of 20ms (0-65535). Specify: `SlavePollTimeout := value`
	EOTSuppression	Specify whether to enable EOT suppression. Enter 0 to disable; enter 1 to enable. Specify: `EOTSuppression := value`
ExtendedDevice	none	none
FaultLog	none	none
ICP	none	none
PCCC	none	none
Redundancy	none	none

Object:	Attribute:	Description:
SerialPort	BaudRate	Specify the communication rate for the serial port. Enter 110, 300 600, 1200, 2400, 4800, 9600, or 19200 Specify: `BaudRate := value`
	Parity	Specify the parity setting for the serial port. Parity provides additional message-packet error detection. Enter None Parity, Even Parity, or Odd Parity. Specify: `Parity := text`
	DataBits	Specify the number of bits per message packet. Enter 0 7 Data Bits or 8 Data Bits. Specify: `DataBits := text`
	StopBits	Specify the number of stop bits to the device with which the controller is communicating. Enter 1 Stop Bit or 2 Stop Bit. Specify: `StopBits := text`
	ComDriverId	Specify the type of serial driver. Enter DF1. Specify: `ComDriverId := text`
	PendingComDriverId	Specify type of serial driver. Enter DF1. Specify: `PendingComDriverId := text`
	RTSOffDelay	Specify a time delay to make sure the modem successfully transmits the entire message. Enter an increment of 20ms (0-32767). Normally leave at zero. Specify: `RTSOffDelay := value`
	RTSSendDelay	Specify a time delay to let the modem prepare to transmit a message. Enter an increment of 20ms (0-32767). Specify: `RTSSendDelay := value`
	ControlLine	Specify the mode in which the serial driver operates. Enter No Handshake, Full Duplex, Half Duplex without Continuous Carrier, or Half Duplex with Continuous Carrier. Specify: `ControlLine := text`
	PendingControlLine	Specify the mode in which the serial driver operates. Enter No Handshake, Full Duplex, Half Duplex without Continuous Carrier, or Half Duplex with Continuous Carrier. Specify: `PendingControlLine := text`
	RemoteModeChangeFlag	Specify whether there is a remote change. Enter 0 or 1. Specify: `RemoteModeChangeFlag := value`
	PendingRemoteModeChange Flag	Specify whether there is a remote change. Enter 0 or 1. Specify: `PendingRemoteModeChangeFlag := value`
	ModeChangeAttentionChar	Specify the mode change attention character. Specify: `ModeChangeAttentionChar := value`
	PendingModeChange AttentionChar	Specify the mode change attention character. Specify: `PendingModeChangeAttentionChar := value`

Object:	Attribute:	Description:
SerialPort (continued)	SystemModeCharacter	Specify the system mode character. Specify: `SystemModeCharacter := value`
	PendingSystemMode Character	Specify the system mode character. Specify: `PendingSystemModeCharacter := value`
	UserModeCharacter	Specify the user mode character. Specify: `SystemModeCharacter := value`
	PendingSystemMode Character	Specify the user mode character. Specify: `PendingSystemModeCharacter := value`
UserMemory	none	none
WallClockTime	LocalTimeAdjustment	Specify any local time adjustment. Specify: `LocalTimeAdjustment := value`
	TimeZone	Specify the time zone. Specify: `TimeZone := value`

CONFIG examples

The following two examples show a DF1 controller object and a SerialPort controller object.

```
CONFIG DF1
       DuplicateDetection := -1,
       ErrorDetection := BCC Error,
       EmbeddedResponseEnable := -1,
       DF1Mode := Pt to Pt,
       ACKTimeout := 50,
       NAKReceiveValue := 3,
       DF1ENQs := 3,
       DF1Retries := 3,
       StationAddress := 0,
       ReplyMessageWait := 50,
       PollingMode := 0,
       MasterMessageTransmit := 0,
       NormalPollNodeFile := NA,
       NormalPollGroupSize := 0,
       PriorityPollNodeFile := NA,
       ActiveStationFile := NA)
END_CONFIG


CONFIG SerialPort
       (BaudRate := 19200,
       Parity := No Parity,
       DataBits := 8 Bits of Data,
       StopBits := 1 Stop Bit,
       ComDriverId := DF1,
       RTSOffDelay := 0,
       RTSSendDelay := 0,
       ControlLine := No Handshake,
       RemoteModeChangeFlag := 0,
       ModeChangeAttentionChar := 27,
       SystemModeCharacter := 83,
       UserModeCharacter := 85)
END_CONFIG
```

Notes:

Entering Ladder Diagram Logic

Introduction

This chapter explains the how to enter ladder diagram logic in a complete import/export file.

For information about:	See page:
Entering rung logic	4-1
Entering comments	4-3
Entering instructions in neutral text language	4-3

Entering Rung Logic

You enter rung logic within a ROUTINE component in an import/export file. Each rung follows this structure:

```
<RungType> : <RungNeutralText>;
```

Where:

Item:	Identifies:
`RungType`	the rung
`RungNeutralText`	the logic

The following rung types are available:

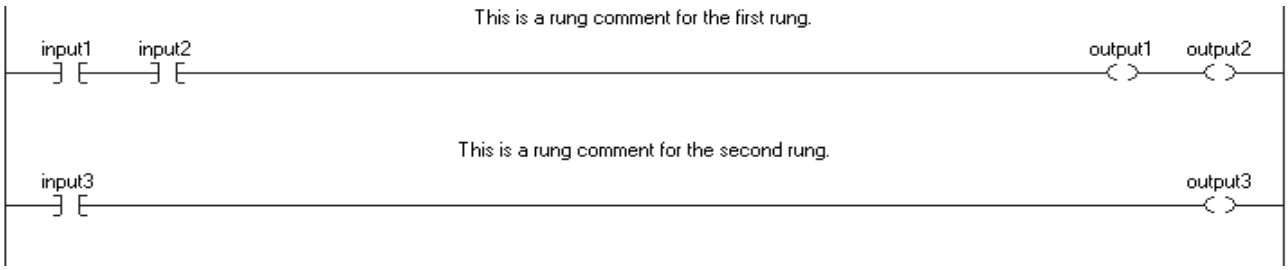
Rung type:	Description:
N	normal
I	insert
D	delete
IR	insert with a replace
rR	pending replace IR
R	replace
rI	pending replace I
rN	pending replace N
e	pending insert rung
er	pending replace rung

Rung guidelines

- Rungs are specified using neutral language. See the rest of this chapter for the neutral text language format for the supported instructions.
- Each rung ends with a semicolon (;).

Ladder ROUTINE example

This ladder routine exports as shown below.




```
ROUTINE Ladder_example
       RC: "This is a rung comment for the first rung.";
       N: XIC(input1)XIC(input2)OTE(output1)OTE(output2);
       RC: "This is a rung comment for the second rung.";
       N: XIC(input3)OTE(output3);
END_ROUTINE
```

Entering Branches

You can enter a single branch or simultaneous branches on a rung. A branch follows this structure:

`[,`*`BranchNeutralText`*`]`

Where:

Item:	Identifies:
[ ]	the branch
,	the beginning of each branch within the branch, to account for simultaneous branches
space	the end of each branch within the branch, to account for simultaneous branches
BranchNeutralText	the logic

Example with a single branch

```
N: XIC(conveyor_a)[,XIC(input_1) XIO(input_2) ]OTE(light_1);
```

Example with two simultaneous branches

```
N: XIC(conveyor_b)[,XIC(input_1) XIO(input_2) ,XIC(input_a) XIO(input_b)  ]OTE(light_2);
```

Entering Rung Comments

The comments for rungs are similar to those for components except that the syntax is a bit different. The rung comment syntax is:

```
RC: “comment” “more” “etc”;
```

A rung comment must be followed by a rung.

Entering Neutral Text for Ladder Instructions

The following tables lists each ladder instruction and its neutral text format. For details about a specific instruction, see one of these manuals:

Instruction Type:	Documents:
Basic, sequential instruction	*Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003
Process control or drives instruction	*Logix5000 Controllers Process Control and Drives Instruction Set Reference Manual*, publication 1756-RM006
Motion instruction	*Logix5000 Controllers Motion Instruction Set Reference Manual,* publication 1756-RM007

Instruction:	Neutral text format:
ABL	`ABL(channel,serial_port_control,character_count);`
ABS	`ABS(source,destination);`
ACB	`ACB(channel,serial_port_control,character_count);`
ACL	`ACL(channel,clear_serial_port_read,clear_serial_port_write);`
ACS	`ACS(source,destination);`
ADD	`ADD(source_A,source_B,destination);`
AFI	`AFI();`
AHL	`AHL(channel,ANDMask,ORMask,serial_port_control,channel_status);`
AND	`AND(source_A,source_B,destination);`
ARD	`ARD(channel,destination,serial_port_control,string_length, characters_read);`

Instruction:	Neutral text format:
ARL	`ARL(channel,destination,serial_port_control,string_length, characters_read);`
ASN	`ASN(source,destination);`
ATN	`ATN(source,destination);`
AVE	`AVE(array,dim_to_vary,destination,control,length,position);`
AWA	`AWA(channel,source,serial_port_control,string_length,characters_sent);`
AWT	`AWT(channel,source,serial_port_control,string_length,characters_sent);`
BRK	`BRK();`
BSL	`BSL(array,control,source_bit,length);`
BSR	`BSR(array,control,source_bit,length);`
BTD	`BTD(source,source_bit,destination,destination_bit,length);`
CLR	`CLR(destination);`
CMP	`CMP(expression);`
CONCAT	`CONCAT (sourceA,sourceB,destination)`
COP	`COP(source,destination,length);`
COS	`COS(source,destination);`
CPS	`CPS (source,destination,length)`
CPT	`CPT(destination,expression);`
CTD	`CTD(counter,preset,accum);`
CTU	`CTU(counter,preset,accum);`
DDT	`DDT(source,reference,result,cmp_control,length,position,result_control, length,position);`
DEG	`DEG(source,destination);`
DELETE	`DELETE (source,quantity,start,destination)`
DIV	`DIV(source_A,source_B,destination);`
DTOS	`DTOS (source,destination)`
DTR	`DTR(source,mask,reference);`
EQU	`EQU(source_A,source_B);`
FAL	`FAL(control,length,position,mode,destination,expression);`
FBC	`FBC(source,reference,result,cmp_control,length,position,result_control, length,position);`
FIND	`FIND (source,search,start,result)`
FFL	`FFL(source,FIFO,control,length,position);`
FFU	`FFU(FIFO,destination,control,length,position);`
FLL	`FLL(source,destination,length);`
FOR	`FOR(routine_name,index,initial_value,terminal_value,step_size);`
FRD	`FRD(source,destination);`
FSC	`FSC(control,length,position,mode,expression);`

Instruction:	Neutral text format:
GEQ	GEQ(*source_A,source_B*);
GRT	GRT(*source_A,source_B*);
GSV	GSV(*object_class,object_name,attribute_name,destination*);
INSERT	INSERT (*sourceA,sourceB,start,destination*)
JMP	JMP(*label_name*);
JSR	JSR(*routine_name,input_1,...input_n,return_1,..return_n*);
LBL	LBL(*label_name*);
LEQ	LEQ(*source_A,source_B*);
LES	LES(*source_A,source_B*);
LFL	LFL(*source,LIFO,control,length,position*);
LFU	LFU(*LIFO,destination,control,length,position*);
LIM	LIM(*low_limit,test,high_limit*);
LN	LN(*source,destination*);
LOG	LOG(*source,destination*);
MAAT	MAAT(*axis,motion_control*);
MAFR	MAFR(*axis,motion_control*);
MAG	MAG(slave*_axis,master_axis,motion_control,direction,ratio,slave_counts,mas ter_counts,master_reference,ratio_format,clutch,accel_rate,units*);
MAH	MAH(*axis,motion_control*);
MAHD	MAHD(*axis,motion_control,test,direction*);
MAJ	MAJ(*axis,motion_control,direction,speed,units,accel_rate,units, decel_rate,units,profile,merge,merge_speed*);
MAM	MAM(*axis,motion_control,move_type,position,speed,units,accel_rate units,decel_rate,units,profile,merge,merge_speed*);
MAOC	MAOC(*axis,execution_target,motion_control,output,input,output_cam, cam_start_position,cam_end_position,output_compensation,execution_mode, execution_schedule,axis_arm_position,cam_arm_position,position_reference*)
MAPC	MAPC(*slave_axis*,*master_axis*,*motion_control*,*direction*,*cam_profile*, *slave_scaling*,*master_scaling*,*execution_mode*,*execution_schedule*, *master_lock_position*,*cam_lock_position*,*master_reference*, *master_direction*);
MAR	MAR(*axis,motion_control,trigger,registration,minimum,maximum*);
MAS	MAS(*axis,motion_control,stop_type,change_decel,rate_units*);
MASD	MASD(*axis,motion_control*);
MASR	MASR(*axis,motion_control*);
MATC	MATC(*axis*,*motion_control*,*direction*,*cam_profile*,*distance_scaling*, *time_scaling*,*execution_mode*,*execution_schedule*);
MAW	MAW(*axis,motion_control,trigger,position*);
MCCP	MCCP(*motion_control*,*cam*,*length*,*start_slope*,*end_slope*,*cam_profile*);

Instruction:	Neutral text format:
MCD	MCD(*axis,motion_control,motion_type,change_speed,speed,change_accel, accel_rate,change_decel,decel_rate,speed_units,accel_units, decel_units*);
MCR	MCR();
MDF	MDF(*axis,motion_control*);
MDO	MDO(*axis,motion_control,drive_output,drive_units*);
MDOC	MDOC(*axis,execution_target,motion_control,disarm_type*)
MDR	MDR(*axis,motion_control*);
MDW	MDW(*axis,motion_control*);
MEQ	MEQ(*source,mask,compare*);
MGPS	MGPS(*group,motion_control*);
MGS	MGS(*group,motion_control,inhibit*);
MGSD	MGSD(*group,motion_control*);
MGSP	MGSP(*group,motion_control*);
MGSR	MGSR(*group,motion_control*);
MID	MID(*source,quantity,start,destination*)
MOD	MOD(*source_A,source_B,destination*);
MOV	MOV(*source,destination*);
MRAT	MRAT(*axis,motion_control*);
MRHD	MRHD(*axis,motion_control,test*);
MRP	MRP(*axis,motion_control,type,position_select,position*);
MSF	MSF(*axis,motion_control*);
MSG	MSG(*message_control*);
MSO	MSO(*axis,motion_control*);
MUL	MUL(*source_A,source_B,destination*);
MVM	MVM(*source,mask,destination*);
NEG	NEG(*source,destination*);
NEQ	NEQ(*source_A,source_B*);
NOP	NOP();
NOT	NOT(*source,destination*);
ONS	ONS(*storage_bit*);
OR	OR(*source_A,source_B,destination*);
OSF	OSF(*storage_bit,output_bit*);
OSR	OSR(*storage_bit,output_bit*);
OTE	OTE(*data_bit*);
OTL	OTL(*data_bit*);
OTU	OTU(*data_bit*);

Instruction:	Neutral text format:
PID	PID(*pv,pv_type,tieback,cv,cv_type,master,inhold_bit,inhold_value*);
RAD	RAD(*source,destination*);
RES	RES(*structure*);
RET	RET(*return_1,...return_n*);
RTO	RTO(*timer,preset,accum*);
RTOS	RTOS(*source,destination*)
SBR	SBR(*routine_name,input_1,...input_n*);
SIN	SIN(*source,destination*);
SQI	SQI(*array,mask,source,control,length,position*);
SQL	SQL(*array,source,control,length,position*);
SQO	SQO(*array,mask,destination,control,length,position*);
SQR	SQR(*source,destination*);
SRT	SRT(*array,dim_to_vary,control,length,position*);
SSV	SSV(*object_class,object_name,attribute_name,destination*);
STD	STD(*array,dim_to_vary,destination,control,length,position*);
STOD	STOD(*source,destination*)
STOR	STOR(*source,destination*)
SUB	SUB(*source_A,source_B,destination*);
TAN	TAN(*source,destination*);
TND	TND();
TOD	TOD(*source,destination*);
TOF	TOF(*timer,preset,accum*);
TON	TON(*timer,preset,accum*);
TRN	TRN(*source,destination*);
UID	UID();
UIE	UIE();
XIC	XIC(*data_bit*);
XIO	XIO(*data_bit*);
XOR	XOR(*source_A,source_B,destination*);
XPY	XPY(*source_A,source_B,destination*);

Notes:

Entering Function Block Diagram Logic

Introduction

This chapter explains the how to enter function block diagram logic in a complete import/export file.

For information about:	See page:
Entering function block diagram logic	5-1
Entering IREFs and OREFs	5-5
Entering ICONs and OCONs	5-7
Entering blocks	5-9
Entering wires	5-8
Entering instructions in neutral text	5-11

Entering Function Block Diagram Logic

You enter function block diagram logic in sheets within a FBD_ROUTINE component in an import/export file. Each sheet follows this structure:

```
(* sheet <sheet_number> *)
SHEET (Name := <sheet_name>)
       <IREF_component>
       <ICON_component>
       <mnemonic_BLOCK_componment>
       <OREF_component>
       <OCON_component>
       <WIRE_component>
       <FEEDBACK_WIRE_component>
END_SHEET
```

Where:

Item:	Identifies:
`Name`	the name of the sheet. Specify: `Name := "text"`
`IREF`	input references see page 5-5
`ICON`	input wire connectors see page 5-7

Item:	Identifies:
mnemonic_BLOCK	function block instructions and their locations see page 5-9
OREF	output references see page 5-5
OCON	output wire connectors see page 5-7
WIRE	wires and what they are attached to see page 5-8
FEEDBACK_WIRE	feedback wires and what they are attached to see page 5-8

SHEET guidelines

- The sheets in the routine appear in order in the export file. Each sheet section contains all the drawing elements and wires for that sheet.
- The sheet number is stored in a comment at the beginning of the sheet for reference only. On import, sheet numbers are assigned based on order in the file, not on the number in the comment.
- The sheet name is stored as an attribute because it is optional.
- Input references, blocks, output references, special drawing elements, and wires are contained within the sheet. On export, the elements appear in the order shown. On import, elements can be interspersed in the file.
- WIRE and FEEDBACK_WIRE statements must appear after all the other components.
- Be careful when copying and pasting function block components within an import/export file. Each component within a sheet must have a unique ID number within that sheet.

FBD_ROUTINE example

```
FBD_ROUTINE My_FBD_Routine (SheetSize := "Tabloid (11x17in)", SheetOrientation := Landscape)
       SHEET  (Name := Input_Scaling)
              MUL_BLOCK  (ID := 0,
                          X := 440,
                          Y := 60,
                          Operand := MUL_01,
                          VisiblePins := "SourceA, SourceB, Dest")
              END_MUL_BLOCK

              SCL_BLOCK  (ID := 1,
                          X := 240,
                          Y := 60,
                          Operand := SCL_01,
                          VisiblePins := "In, InEUMax, Out, MaxAlarm")
              END_SCL_BLOCK

              PI_BLOCK  (ID := 2,
                         X := 260,
                         Y := 260,
                         Operand := PI_01,
                         VisiblePins := "In, Initialize, InitialValue, Out, HighAlarm,
LowAlarm")
              END_PI_BLOCK

              IREF  (ID := 3,
                     X := 120,
                     Y := 120,
                     Operand := Input_Tag)
              END_IREF

              ICON  (ID := 4,
                     X := 160,
                     Y := 320,
                     Name := ConnectorName)
              END_ICON
```

```
OREF  (ID := 5,
       X := 520,
       Y := 320,
       Operand := Output_Tag)
END_OREF

OCON  (ID := 6,
       X := 680,
       Y := 100,
       Name := ConnectorName)
END_OCON

FEEDBACK_WIRE  (FromElementID := 0,
                FromParameter := Dest,
                ToElementID := 0,
                ToParameter := SourceB)
END_FEEDBACK_WIRE

WIRE  (FromElementID := 3,
       FromParameter := "",
       ToElementID := 1,
       ToParameter := In)
END_WIRE

WIRE  (FromElementID := 4,
       FromParameter := "",
       ToElementID := 2,
       ToParameter := In)
END_WIRE

WIRE  (FromElementID := 0,
       FromParameter := Dest,
       ToElementID := 6,
       ToParameter := "")
END_WIRE
```

```
            WIRE  (FromElementID := 1,
                   FromParameter := Out,
                   ToElementID := 0,
                   ToParameter := SourceA)
            END_WIRE

            WIRE  (FromElementID := 2,
                   FromParameter := Out,
                   ToElementID := 5,
                   ToParameter := "")
            END_WIRE

      END_SHEET

END_FBD_ROUTINE
```

Entering IREFs and OREFs

Input and output references have similar formats and identical attributes. They follow this format:

```
IREF (
       ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Operand := <tag_reference>)
END_IREF

OREF (
       ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Operand := <tag_reference>)
END_OREF
```

Where:

Item:	Identifies:
ID	the IREF or OREF identifier; uniqueness is important for wiring enter unsigned, 32-bit integer value Specify: `ID := "number"`
X	x-coordinates on internal grid enter unsigned, 32-bit integer value Specify: `X := "number"`
Y	y-coordinates on internal grid enter unsigned, 32-bit integer value Specify: `Y := "number"`
Operand	the reference (optional) enter tag or literal value for IREF; enter tag for OREF Specify: `Operand := "tag"`

IREF and OREF guidelines

- If the Operand is not a qualified tag or literal value, the IREF/OREF will not be verified.
- The X and Y grid locations are a relative position from the upper-left corner of the sheet. X is the horizontal position; Y is the vertical position.

IREF and OREF examples

```
IREF  (ID := 8,
       X := 200,
       Y := 380,
       Operand := PMUL_InitVal)
END_IREF


OREF  (ID := 9,
       X := 480,
       Y := 340,
       Operand := FB_PMUL)
END_OREF
```

Entering ICONs and OCONs

Input and output wire connectors have similar formats and identical attributes. They follow this format:

```
ICON (ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Name := <connector_name>)
END_ICON

OCON (ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Name := <connector_name>)
END_OCON
```

Where:

Item:	Identifies:
ID	the ICON or OCON identifier; uniqueness is important for wiring enter unsigned, 32-bit integer value Specify: `ID := "number"`
X	x-coordinates on internal grid enter unsigned, 32-bit integer value Specify: `X := "number"`
Y	y-coordinates on internal grid enter unsigned, 32-bit integer value Specify: `Y := "number"`
Name	the name of the wire connector (optional) Specify: `Name := "number"`

ICON and OCON guidelines

- OCON connector names must be unique within a function block routine.
- Multiple ICON connector names can reference the same OCON connector name.
- ICONs and OCONs with unmatched or blank connector names will not be verified.
- The X and Y grid locations are a relative position from the upper-left corner of the sheet. X is the horizontal position; Y is the vertical position.

ICON and OCON examples

```
ICON  (ID := 1,
       X := 140,
       Y := 300,
       Name := MyConnector)
END_ICON

OCON  (ID := 4,
       X := 460,
       Y := 140,
       Name := MyConnector)
END_OCON
```

Entering Wires and Feedback Wires

The wire and feedback wire formats describe a wire by specifying what it is attached to at each end, which is always a pin on another drawing element. Wires and feedback wires follow this format:

```
WIRE
       FromElementID := <identifier_of_from_element>,
       FromParameter := <name_of_output_pin>,
       ToElementID := <identifier_of_to_element>,
       ToParameter := <name_of_input_pin>
END_WIRE
```

Where:

Item:	Identifies:
`FromElementID`	the source drawing element enter unsigned, 32-bit integer Specify: `FromElementID := "number"`
`FromParameter`	the pin on the source drawing element **For:** **Enter:** blocks parameter name IREFs In ICONs In Specify: `FromParameter := "pin"`
`ToElementID`	the destination drawing element enter unsigned, 32-bit integer Specify: `ToElementID := "number"`
`ToParameter`	the pin on the destination drawing element **For:** **Enter:** blocks parameter name OREFs Out OCONs Out Specify: `ToParameter := "pin"`

WIRE guidelines

- Wires that are not correctly specified will not be imported.
- A feedback wire follows the same format as a wire. Just connect the source and destination elements to form a feedback.

WIRE example

```
WIRE  (FromElementID := 11,
       FromParameter := "",
       ToElementID := 1,
       ToParameter := Initialize)
END_WIRE
```

Entering Blocks

All function blocks follow this format:

*mnemonic*_BLOCK (
ID := <*unique_identifier*>
X := <*internal_grid_x_location*>,
Y := <*internal_grid_y_location*>,
Operand := <*block_tag_reference*>,
<*Array_Name*>Operand := <*array_tag_reference*>,
VisiblePins := "<*parameter_name*>, …")
END_*mnemonic*_BLOCK

Where:

Item:	Identifies:
ID	the block identifier; uniqueness is important for wiring enter unsigned, 32-bit integer value Specify: ID := *"number"*
X	x-coordinates on internal grid enter unsigned, 32-bit integer value Specify: X := *"number"*
Y	y-coordinates on internal grid enter unsigned, 32-bit integer value Specify: Y := *"number"*

Item:	Identifies:
`Operand`	tag name for the block (optional) Specify: `Operand := "tag_name"`
`ArrayName`	tag name for array (optional) Specify: `ArrayName := "array_name"`
`VisiblePins`	Comma-separated list of the names of all the parameters with pins visible for wiring. The names match the member names of the data type of the block tag. Specify: `VisiblePins := "parameter"`

BLOCK guidelines

- If the Operand is not a qualified tag of the correct data type, the block will not be verified.
- Some function block instructions require specific arrays. This table lists the valid Array Name for each of these instructions:

Instruction:	Array Name:
DEDT	Storage (required)
FGEN	X1 (required) Y1 (required) X2 (optional) Y2 (optional)
MAVE	Storage (required) Weight (optional)
RMPS	RampValue (required) SoakValue (required) SoakTime (required)

- The X and Y grid locations are a relative position from the upper-left corner of the sheet. X is the horizontal position; Y is the vertical position.

See page 5-11 for details on the parameters for the specific function block instructions.

Entering Parameters for Function Block Instructions

The following tables lists each function block instruction and its format in the Block component of an import/export file. For details about a specific instruction, see one of these manuals:

Instruction Type:	Documents:
Basic, sequential instruction	*Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003
Process control or drives instruction	*Logix5000 Controllers Process Control and Drives Instruction Set Reference Manual*, publication 1756-RM006
Motion instruction	*Logix5000 Controllers Motion Instruction Set Reference Manual,* publication 1756-RM007

Instruction:	Default Operand and VisiblePins formats (components within the Block structure):
ABS	`Operand := ABS_01,` `VisiblePins := "Source, Destination")`
ACS	`Operand := ACS_01,` `VisiblePins := "Source, Destination")`
ADD	`Operand := ADD_01,` `VisiblePins := "SourceA, SourceB, Destination")`
ALM	`Operand := ALM_01,` `VisiblePins := "In, HHAlarm, HAlarm, LAlarm, LLAlarm, ROCPosAlarm, ROCNegAlarm")`
AND	`Operand := AND_01,` `VisiblePins := "SourceA, SourceB, Destination")`
ASN	`Operand := ASN_01,` `VisiblePins := "Source, Destination")`
ATN	`Operand := ATN_01,` `VisiblePins := "Source, Destination")`
BAND	`Operand := BAND_01,` `VisiblePins := "In1, In2, In3, In4, Out")`
BNOT	`Operand := BNOT_01,` `VisiblePins := "In, Out")`
BOR	`Operand := BOR_01,` `VisiblePins := "In1, In2, In3, In4, Out")`
BTDT	`Operand := BTDT_01,` `VisiblePins := "Source, SourceBit, Length, DestBit, Target, Dest")`
BXOR	`Operand := BXOR_01,` `VisiblePins := "In1, In2, Out")`
COS	`Operand := COS_01,` `VisiblePins := "Source, Dest")`
CTUD	`Operand := CTUD_01,` `VisiblePins := "CUEnable, CDEnable, PRE, Reset, ACC, DN")`

Instruction:	Default Operand and VisiblePins formats (components within the Block structure):
D2SD	Operand := D2SD_01, VisiblePins := "*ProgCommand, State0Perm, State1Perm, FB0, FB1, HandFB, ProgProgReq, ProgOperReq, ProgOverrideReq, ProgHandReq, Out, Device0State, Device1State, CommandStatus, FaultAlarm, ModeAlarm, ProgOper, Override, Hand*")
D3SD	Operand := D3SD_01, VisiblePins := "*Prog0Command, Prog1Command, Prog2Command, State0Perm, State1Perm, State2Perm, FB0, FB1, FB2, FB3, HandFB0, HandFB1, HandFB2, ProgProgReq, ProgOperReq, ProgOverrideReq, ProgHandReq, Out0, Out1, Out2, Device0State, Device1State, Device2State, Command0Status, Command1Status, Command2Status, FaultAlarm, ModeAlarm, ProgOper, Override, Hand*")
DEDT	Operand := DEDT_01, VisiblePins := "*In, Out*", Storage := *array_name*)
DEG	Operand := DEG_01, VisiblePins := "*Source, Dest*")
DERV	Operand := DERV_01, VisiblePins := "*In, ByPass, Out*")
DFF	Operand := DFF_01, VisiblePins := "*D, Clear, Clock, Q, QNot*")
DIV	Operand := DIV_01, VisiblePins := "*SourceA, SourceB, Dest*")
ESEL	Operand := ESEL_01, VisiblePins := "*In1, In2, In3, In4, In5, In6, ProgSelector, ProgProgReq, ProgOperReq, ProgOverrideReq, Out, SelectedIn, ProgOper, Override*")
EQU	Operand := EQU_01, VisiblePins := "*SourceA, SourceB*")
FGEN	Operand := FGEN_01, VisiblePins := "*In, Out*", X1 := *array_name*, X2 := *array_name*, Y2 := *array_name*, Y2 := *array_name*)
FRD	Operand := FRD_01, VisiblePins := "*Source, Dest*")
GEQ	Operand := GEQ_01, VisiblePins := "*SourceA, SourceB*")
GRT	Operand := GRT_01, VisiblePins := "*SourceA, SourceB*")
HLL	Operand := HLL_01, VisiblePins := "*In, Out, HighAlarm, LowAlarm*")

Instruction:	Default Operand and VisiblePins formats (components within the Block structure):
HPF	Operand := HPF_01, VisiblePins := "*In, Out*")
INTG	Operand := INTG_01, VisiblePins := "*In, Out*")
JKFF	Operand := JKFF_01, VisiblePins := "*Clear, Clock, Q, QNot*")
LEQ	Operand := LEQ_01, VisiblePins := "*SourceA, SourceB*")
LES	Operand := LES_01, VisiblePins := "*SourceA, SourceB*")
LIM	Operand := LIM_01, VisiblePins := "*LowLlimit, Test, HighLimit*")
LN	Operand := LN_01, VisiblePins := "*Source, Dest*")
LOG	Operand := LOG_01, VisiblePins := "*Source, Dest*")
LPF	Operand := LPF_01, VisiblePins := "*In, Out*")
MAVE	Operand := MAVE_01, VisiblePins := "*In, Out*", Storage := *array_name*, Weight := *array_name*)
MAXC	Operand := MAXC_01, VisiblePins := "*In, Reset, ResetValue, Out*")
MEQ	Operand := MEQ_01, VisiblePins := "*Source, Mask, Compare*")
MINC	Operand := MINC_01, VisiblePins := "*In, Reset, ResetValue, Out*")
MOD	Operand := MOD_01, VisiblePins := "*SourceA, SourceB, Dest*")
MSTD	Operand := MSTD_01, VisiblePins := "*In, SampleEnable, Out*", Storage := *array_name*)
MUL	Operand := MUL_01, VisiblePins := "*SourceA, SourceB, Dest*")
MUX	Operand := MUX_01, VisiblePins := "*In1, In2, In3, In4, In5, In6, In7, In8, Selector, Out*")
MVMT	Operand := MVMT_01, VisiblePins := "*Source, Mask, Target, Dest*")

Instruction:	Default Operand and VisiblePins formats (components within the Block structure):
NEG	Operand := NEG_01, VisiblePins := "*Source, Dest*")
NEQ	Operand := NEQ_01, VisiblePins := "*SourceA, SourceB*")
NOT	Operand := NOT_01, VisiblePins := "*Source, Dest*")
NTCH	Operand := NTCH_01, VisiblePins := "*In, Out*")
OR	Operand := OR_01, VisiblePins := "*SourceA, SourceB, Dest*")
OSFI	Operand := OSFI_01, VisiblePins := "*InputBit, OutputBit*")
OSRI	Operand := OSRI_01, VisiblePins := "*InputBit, OutputBit*")
PI	Operand := PI_01, VisiblePins := "*In, Out*")
PIDE	Operand := PIDE_01, VisiblePins := "*PV, SPProg, SPCascade, RatioProg, CVProg, FF, HandFB, ProgProgReq, ProgOperReq, ProgCasRatReq, ProgAutoReq, ProgManuaReq, ProgOverrideReq, ProgHandReq, CVEU, SP, PVHHAlarm, PVHAlarm, PVLAlarm, PVLLAlarm, PVROCPosAlarm, PVROCNegAlarm, DevHHAlarm, DevHAlarm, DevLAlarm, DevLLAlarm, ProgOper, CasRat, Auto, Manual, Override, Hand*")
PMUL	Operand := PMUL_01, VisiblePins := "*In, Multipler, Out*")
POSP	Operand := POSP_01, VisiblePins := "*SP, Position, OpenedFB, ClosedFB, OpenOut, CloseOut*")
RAD	Operand := RAD_01, VisiblePins := "*Source, Dest*")
RESD	Operand := RESD_01, VisiblePins := "*Set, Reset, Out, OutNot*")
RLIM	Operand := RLIM_01, VisiblePins := "*In, ByPass, Out*")
RMPS	Operand := RMPS_01, VisiblePins := "*PV, CurrentSegProg, OutProg, SoakTimeProg, ProgProgReq, ProgOperReq, ProgAutoReq, ProgManualReq, ProgHoldReq, Out, CurrentSeg, SoakTimeLeft, GuarRampOn, GuarSoakOn, ProgOper, Auto, Manual, Hold*", RampValue := *array_name*, SoakValue := *array_name*, SoakTime := *array_name*)
RTOR	Operand := RTOR_01, VisiblePins := *TimerEnable, PRE, Reset, ACC, DN")*

Instruction:	Default Operand and VisiblePins formats (components within the Block structure):
SCL	Operand := SCL_01, VisiblePins := "*In, Out*")
SCRV	Operand := SCRV_01, VisiblePins := "*In, Out*")
SEL	Operand := SEL_01, VisiblePins := "*In1, In2, SelectorIn, Out*")
SETD	Operand := SETD_01, VisiblePins := "*Set, Reset, Out, OutNot*")
SIN	Operand := SIN_01, VisiblePins := SIN(*source,destination*);
SNEG	Operand := SNEG_01, VisiblePins := "*In, NegateEnable, Out*")
SOC	Operand := SOC_01, VisiblePins := "*In, Out*")
SQR	Operand := SQR_01, VisiblePins := "*Source, Dest"*)
SRTP	Operand := SRTP_01, VisiblePins := "*In, HeatOut, CoolOut, HeatTimePercent, CoolTimePercent*")
SSUM	Operand := SSUM_01, VisiblePins := "*In1, Select1, In2, Select2, In3, Select3, In4, Select4, Out*")
SUB	Operand := SUB_01, VisiblePins := "*SourceA, SourceB, Dest"*)
TAN	Operand := TAN_01, VisiblePins := "*Source, Dest")*
TOD	Operand := TOD_01, VisiblePins := "*Source, Dest")*
TOFR	Operand := TOFR_01, VisiblePins := "*TimerEnable, PRE, Reset, ACC, DN*")
TONR	Operand := TONR_01, VisiblePins := "*TimerEnable, PRE, Reset, ACC, DN*")
TOT	Operand := TOT_01, VisiblePins := "*In, ProgProgReq, ProgOperReq, ProgStartReq, ProgStopReq, ProgResetReq, Total, OldTotal, ProgOper, RunStop, ProgResetDone, TargetFlag, TargetDev1Flag, TargetDev2Flag*")
TRN	Operand := TRN_01, VisiblePins := "*Source, Dest")*

Instruction:	Default Operand and VisiblePins formats (components within the Block structure):
UPDN	`Operand := UPDN_01,` `VisiblePins := "InPlus, InMinus, Out")`
XOR	`Operand := XOR_01,` `VisiblePins := "SourceA, SourceB, Dest")`
XPY	`Operand := XPY_01,` `VisiblePins := "SourceA, SourceB, Dest")`

Structuring the Tag (.CSV) Import/Export File Format

Introduction

This chapter explains the overall structure of a tag import/export file. The file extension for a tag import/export file is .CSV.

IMPORTANT To edit the .CSV file, it is recommended that you use a database program tool, such as Microsoft Access, or a raw text editor. Many other desktop tools, such as Microsoft Word or Excel, might change the structure of the .CSV file and cause an import of the file to fail. For more information on the implications of using Excel to edit the exported .CSV file, see appendix A.

Conventions

The tag import/export utility is based on the CSV format used by spreadsheet programs. The examples follow these conventions:

Convention:	Meaning:
user_value	items in italics indicate user-supplied information
LITERAL	items in all uppercase indicate a required keyword or symbol that must be entered as shown
" "	double quotes must enclose some values, as shown in the examples

The CSV (comma separated values) format uses the separator value defined in the registry to identify separate information. A common separator value is a comma. In Windows NT, check your separator value, or change the value, with the List separator option on the Number tab at:

Start → Settings → Control Panel → Regional Settings

White space characters include spaces, tabs, carriage return, newline, and form feed. These characters can occur anywhere in an import/export file, except in keywords or names. If white space characters occur outside of descriptions, they are ignored.

Internal file comments

You can enter comments to document your import files. The import process ignores these comments. You can place comments anywhere in an import/export file, except in names and descriptions. You enter comments by starting the line (record) with REMARK and a comma.

Placing Information in a Tag (.CSV) Import/Export File

The tag import/export file contains two components of information. These components are:

Item:	Identifies:
header information	the content of the tag import/export file each line is a comment line
record	each tag is an individual record

The overall format is:

```
remark,"CSV-Import-Export"
remark,"Date = Fri Mar 16 14:16:47 2001"
remark,"Version = RSLogix 5000-8.00.00.00"
remark,"Owner = User Name"
remark,"Company = Allen-Bradley Co., Inc."
0.1
TYPE,SCOPE,NAME,DESCRIPTION,DATATYPE,SPECIFIER
remark Controller Tags
TAG
.
.
ALIAS
.
.
TYPE,SCOPE,NAME,DESCRIPTION,DATATYPE,SPECIFIER
remark 1st program
TAG
.
.
remark 1st program
ALIAS
.
.
remark last program
```

```
TAG
.
.
remark last program
ALIAS
.
.
```

Global tags precede program tags.

All records in a tag import/export file follow this structure:

```
Type,"Scope","Name","Description","Datatype","Specifier"
```

Where:

Item:	Identifies:
`Type`	the type of tag valid types are: TAG tag ALIAS alias tag COMMENT tag operand component
`Scope`	what part of the project owns the tag if no scope is specified, the scope is controller if a scope is specified, it identifies the program
`Name`	name of the tag
`Description`	description of the tag (optional) enclose in double quotes
Datatype	datatype of the tag - use any valid datatype name
Specifier	optional • for an alias, specifies base tag • for a tag comment, specifies the tag name and member or bit

Specifying a Tag Record

Each TAG record defines a tag within a controller project. A TAG record follows this format:

```
TAG,"Scope","Name","Description","Datatype","Specifier"
```

Specifying dimensions

You specify tag dimensions the same way as you enter the tag in logic.

To specify:	Enter:
1 dimension	number
2 dimensions	number,number
3 dimensions	number,number,number

TAG examples

The following examples show TAG records.

Example:	Description:
TAG,,timer_1,"this is the first timer","TIMER",""	There is no scope specified, so this is a controller scoped tag. The tag is named timer_1. The description is "this is the first timer." The datatype is TIMER There are no additional specifiers for this tag.
TAG,,fault_record,"",fault_structure,""	There is no scope specified, so this is a controller scoped tag. The tag is named fault_record. There is no description. The datatype is a user-defined fault_structure. There are no additional specifiers for this tag.
TAG,recipe_b,int_array,"",INT[10,10],""	This tag is program-scoped to the program named recipe_b. The tag is named int_array. There is no description. The datatype is INT[10,10] - a INT array with two dimensions. There are no additional specifiers for this tag.

Specifying an Alias Record

Each ALIAS record defines an alias within a controller project. An ALIAS record follows this format:

```
ALIAS,"Scope","Name","Description","Datatype","Specifier"
```

The following examples show ALIAS records.

Example:	Description:
ALIAS,,hot,"","","temp"	There is no scope specified, so this is a controller scoped tag. The alias tag is named hot. There is no description. There is no datatype for an alias, it is the same as the base tag. The specifier is the name of the base tag.
ALIAS,recipe_b,start_value,"have this much at first","","int_a"	This tag is program-scoped to the program named recipe_b. The alias tag is names start_value. The description is "have this much at first." There is no datatype for an alias, it is the same as the base tag. The specifier is the name of the base tag.

Specifying a Comment Record

Each COMMENT record defines a comment about a component of a TAG record. A COMMENT record follows this format:

```
COMMENT,"Scope","Name","Description","Datatype","Specifier"
```

The following examples show COMMENT records.

Example:	Description:
COMMENT,,timer_1,"this is the enable bit",,"timer_1.en"	There is no scope specified, so this is a controller scoped tag. The comment is associated with the tag "timer_1". The description is "this is the enable bit". There is no datatype for a comment. The specifier is the tag member associated with the comment.
COMMENT,,ratio,"this is the bit to monitor",,"ratio.3"	There is no scope specified, so this is a controller scoped tag. The comment is associated with the tag "ratio". The description is "this is the bit to monitor." There is no datatype for a comment. The specifier is the bit associated with the comment.
COMMENT,recipe_b,table,"look at this element",,"table[8]"	This tag is program-scoped to the program named recipe_b. The comment is associated with the tag "table". The description is "look at this element." There is no datatype for a comment. The specifier is the element of the array associated with the comment.
COMMENT,,mask_1,"use this mask value",,""	There is no scope specified, so this is a controller scoped tag. The comment is associated with the tag "mask_1". The description is "use this mask value." There is no datatype for a comment. There is no specifier because this specifies a description of a tag, not a component of a tag. **Important:** If you use the COMMENT statement for a tag, it overwrites the description part of the TAG statement for that tag. The COMMENT tag essentially defines a new description for the tag.

Sample Scenarios of Importing/Exporting Tags

The following examples show how you can use the partial import of tags (the collision mode is overwrite):

Scenario:	Result:
Export tags Edit tag attributes, but not names Import tags back into controller project	Changed attributes overwrite existing tag attributes
Export tags (contains tag named Joe) Open .CSV file in Excel Change Joe to Joseph Close file Import file back into controller project	Both Joe and Joseph are in the tag list Any logic referring to Joe still refers to Joe
Export tags Add new tags Import tags back into controller project	New tags are added
Export tags Delete tags Sam and Mitch Import tags back into controller project	Tags Sam and Mitch still exist in tag list

If you have tags in the import file that do not exist in the open project, you can select whether to create these tags in the project. If you select to create these tags, keep in mind:

- If you are importing just a description of a tag, the import process does not create a tag for that description. The COMMENT record in a .CSV file does not contain enough information to create an associated tag. There must be a TAG or ALIAS record to create a tag.

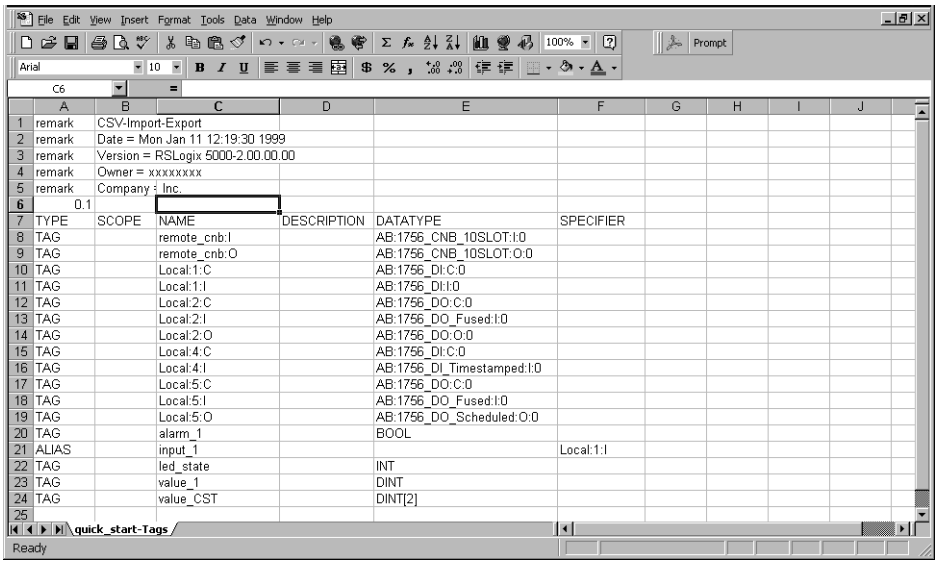
- For a COMMENT record to import correctly, the associated tag has to exist in the RSLogix 5000 database. The tag can either already exist before the import or be created due to a TAG or ALIAS record that created the tag during the import before the COMMENT record is encountered.

Example Tag Import/Export File

The following example shows a partial, tag (.CSV) import/export file.

```
remark,”CSV-Import-Export”
remark,”Date = Fri Mar 16 14:16:47 2001”
remark,”Version = RSLogix 5000-8.00.00.00”
remark,”Owner = User Name”
remark,”Company = Allen-Bradley Co., Inc.”
0.1
TYPE,SCOPE,NAME,DESCRIPTION,DATATYPE,SPECIFIER
TAG,,remote_cnb:I,"","AB:1756_CNB_10SLOT:I:0",""
TAG,,remote_cnb:O,"","AB:1756_CNB_10SLOT:O:0",""
TAG,,Local:1:C,"","AB:1756_DI:C:0",""
TAG,,Local:1:I,"","AB:1756_DI:I:0",""
TAG,,Local:2:C,"","AB:1756_DO:C:0",""
TAG,,Local:2:I,"","AB:1756_DO_Fused:I:0",""
TAG,,Local:2:O,"","AB:1756_DO:O:0",""
TAG,,Local:4:C,"","AB:1756_DI:C:0",""
TAG,,Local:4:I,"","AB:1756_DI_Timestamped:I:0",""
TAG,,Local:5:C,"","AB:1756_DO:C:0",""
TAG,,Local:5:I,"","AB:1756_DO_Fused:I:0",""
TAG,,Local:5:O,"","AB:1756_DO_Scheduled:O:0",""
TAG,,alarm_1,"","BOOL",""
ALIAS,,input_1,"","","Local:1:I"
TAG,,led_state,"","INT",""
TAG,,value_1,"","DINT",""
TAG,,value_CST,"","DINT[2]",""
```

The above example tag file would look like this if opened using a spreadsheet program:



	A	B	C	D	E	F	G	H	I	J
1	remark	CSV-Import-Export								
2	remark	Date = Mon Jan 11 12:19:30 1999								
3	remark	Version = RSLogix 5000-2.00.00.00								
4	remark	Owner = xxxxxxxx								
5	remark	Company = Inc.								
6	0.1									
7	TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER				
8	TAG		remote_cnb:I		AB:1756_CNB_10SLOT:I:0					
9	TAG		remote_cnb:O		AB:1756_CNB_10SLOT:O:0					
10	TAG		Local:1:C		AB:1756_DI:C:0					
11	TAG		Local:1:I		AB:1756_DI:I:0					
12	TAG		Local:2:C		AB:1756_DO:C:0					
13	TAG		Local:2:I		AB:1756_DO_Fused:I:0					
14	TAG		Local:2:O		AB:1756_DO:O:0					
15	TAG		Local:4:C		AB:1756_DI:C:0					
16	TAG		Local:4:I		AB:1756_DI_Timestamped:I:0					
17	TAG		Local:5:C		AB:1756_DO:C:0					
18	TAG		Local:5:I		AB:1756_DO_Fused:I:0					
19	TAG		Local:5:O		AB:1756_DO_Scheduled:O:0					
20	TAG		alarm_1		BOOL					
21	ALIAS		input_1			Local:1:I				
22	TAG		led_state		INT					
23	TAG		value_1		DINT					
24	TAG		value_CST		DINT[2]					
25										

Considerations for Using Microsoft Excel to Edit a .CSV File

Introduction

This appendix describes how using Microsoft Excel can affect a .CSV file.

IMPORTANT To edit the .CSV file, it is recommended that you use a database program tool, such as Microsoft Access, or a raw text editor. Many other desktop tools, such as Microsoft Word or Excel, might change the structure of the .CSV file and cause an import of the file to fail.

Recommendations

If you use Microsoft Excel to edit your .CSV tag file:

- Use single quotes instead of double quotes within descriptions and comments.
- Do not create descriptions or comments that consist only of numbers, have leading zeros, or have a leading symbol that Microsoft Excel treats specially. For example, do not create descriptions like:

  002
  +2
  =2
  -2
  .0

- Do not create descriptions or comments that start with a +, -, or = symbol. Even if you add text after the symbol, Excel displays `#NAME?` in the cell.

RSLogix 5000 Data Transformations

When RSLogix5000 programming software exports tags, it performs these conversions:

Original content:	Content in .CSV file after export:
‘	$’
“	$Q
newline	NL
tab	$T
$	$$

Microsoft Excel Data Transformation

When you open the exported .CSV file in Excel, these conversions occur:

Original content:	Content in .CSV file after export:	Content after opening in Excel:	Content after saving from Excel:	Details:
.0	".0"	0	0	RSLogix5000 addresses this as the specifier for a tag. If you enter this as an entire comment, you lose any preceding period (.). If you enter any text before or after this, Excel maintains the content.
=2	"=2"	2	2	If you enter this as an entire comment, you lose any preceding equal sign (=). If you enter any text before or after this, Excel maintains the content.
+2	"+2"	2	2	If you enter this as an entire comment, you lose any preceding plus sign (+). If you enter any text before or after this, Excel maintains the content.
002	"002"	2	2	If you enter this as an entire comment, you lose any preceding zeros. If you enter any text before or after this, Excel maintains the content.
test string	"test string"	test string	test string	Excel puts quotes around cell contents only if there is an embedded comma. RSLogix5000 always places double quotes around text. But RSLogix5000 can still handle the description without quotes.
"test string"	"$"test string$""	$test string$""	"$test string$"""""	Both Excel and RSLogix5000 alter content when it includes a dollar sign ($).
has "quoted text" within string	"has $"quoted text$" within string"	has $quoted text$" within string"	"has $quoted text $"" within string"""	Both Excel and RSLogix5000 alter content when it includes a dollar sign ($).
this has ‘embedded’ text	this has $‘embedded$’ text	this has $‘embedded$’ text	this has $‘embedded$’ text	Single quotes work fine in both software packages.

Original content:	Content in .CSV file after export:	Content after opening in Excel:	Content after saving from Excel:	Details:
+text	"+text"	#NAME?	#NAME?	Do not start a description or comment with a plus sign (+).
-text	"-text"	#NAME?	#NAME?	Do not start a description or comment with a minus sign (-).
=text	"=text"	#NAME?	#NAME?	Do not start a description or comment with an equal sign (=).

Notes:

Import/Export Revision History

Introduction

This appendix contains a history of enhancements made to the import/export utility **since** version 1.1 (major revision 1, minor revision 1) that was included with RSLogix 5000 programming software, version 8.0.

For information about:	See page:
backward compatibility	B-1
motion changes to support the SERCOS protocol	B-2
import/export utility version 1.1 (RSLogix programming software version 8.0)	B-6

Backward Compatibility

The import/export utility supports backward compatibility for import operations. This means that the RSLogix 5000 programming software can import .L5K files that were generated by a previous version of the programming software. In some cases, an older .L5K file might not correctly import into newer version of the programming software. The revision history in this appendix will list any conditions when backward compatibility for an import operation does not work as expected.

The import/export utility does not support backward compatibility for export operations. This means that older version of the RSLogix 5000 programming software cannot read .L5K files that were created with newer versions of the programming software.

Each version of the RSLogix 5000 programming software exports .L5K files with a specific import/export version number. The RSLogix 5000 programming software imports any .L5K file with the same major revision number and the same or lower minor revision number. The major revision number increments when there are conditions such that the programming software cannot support backward compatibility for import operations. The minor revision number increments whenever there is a change in the file (a new module, an attribute is added, the set of options for an attribute is changed, etc.) that does not affect backward compatibility for import operations.

IMPORTANT Be careful when copying and pasting between different versions of .L5K files. Do not paste objects from an older .L5K file into a newer version.

Motion Changes to Support the SERCOS Protocol

Version 2.0 (major revision 2, minor revision 0) of the import/export utility that is included with RSLogix 5000 programming software, version 9.0 made significant changes to motion-related tags to support the SERCOS protocol:

- CoarseUpdatePeriod and AutoTagUpdate parameters were added to the MOTION_GROUP tag to support SERCOS (see page 3-14). For reference, the previous structure is described below (page B-3).

- Earlier versions of the import/export utility supported one AXIS tag. To support SERCOS, the import/export utility replaced AXIS with four different axis tags: AXIS_CONSUMED, AXIS_SERVO, AXIS_SERVO_DRIVE, and AXIS_VIRTUAL (see pages 3-16 through 3-22). The previous AXIS tag is incorporated into these new tags, but no longer exists as its own tag. For reference, the AXIS structure is described below (page B-3).

  If you have a version 8.0 import/export file with AXIS tags that you import into version 9.0 software (after changing the import/export version line to 2.0), the AXIS tags convert to:

If the AXIS type is:	It converts to:
unused	AXIS_SERVO
position only	AXIS_SERVO
servo	AXIS_SERVO
consumed	AXIS_CONSUMED
virtual	AXIS_VIRTUAL

MOTION_GROUP tag structure (version 1.1)

Attribute:	Description:
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Comment`	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`GroupType`	Specify the type of motion group, such as Independent. Specify: `GroupType := text`
`CoarseUpdateMultiplier`	Specify the coarse update rate (5-320ms). Specify: `CoarseUpdateMultiplier := value`
`ServoUpdatePeriod`	Specify the servo update period in milliseconds (any positive number) Specify: `ServoUpdatePeriod := value`
`PhaseShift`	Specify the phase shift (0-65,535). Specify: `PhaseShift := value`
`GeneralFaultType`	Specify whether an error generates a major fault or a non-major fault. Enter "Major Fault" or "Non Major Fault." Specify: `GeneralFaultType := text`

AXIS tag structure (version 1.1)

Attribute:	Description:
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Comment`	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MotionGroup`	Enter the name of the associated motion group, or enter NA. Specify: `MotionGroup := text`
`MotionModule`	Enter the name of the associated motion module, or enter NA. Specify: `MotionModule := text`
`AxisState`	Enter Axis-Ready, Direct Drive Control, Servo Control, Axis Faulted, or Axis Shutdown. Specify: `AxisState := text`
`PositionUnits`	Specify the type of units. Specify: `PositionUnits := text`
`TimeUnits`	Enter Seconds or Minutes. Specify: `TimeUnits := text`

Attribute:	Description:
`InstructionSpeedUnits`	Enter Percentage or Engineering Units. Specify: `InstructionSpeedUnits := text`
`InstructionAccelDecelUnits`	Enter Percentage or Engineering Units. Specify: `InstructionAccelDecelUnits := text`
`InstructionMoveProfile`	Enter Trapezoidal or S-Curve. Specify: `InstructionMoveProfile := text`
`InstructionJogProfile`	Specify Trapezoidal or S-Curve. Specify: `InstructionJogProfile := text`
`ConversionConstant`	Specify the conversion constant. Enter a real number from 1.0 to $1.0e^9$. Specify: `ConversionConstant := value`
`HomeMode`	Enter Passive or Active. Specify: `HomeMode := text`
`HomeSequenceType`	Enter Immediate Home, Home To Switch, Home To Marker Only, or Home To Switch With Marker. Specify: `HomeSequenceType := text`
`HomePosition`	Specify the home position (any positive number). Specify: `HomePosition := value`
`HomeSpeed`	Specify the home speed (any positive number). Specify: `HomeSpeed := value`
`HomeReturnSpeed`	Specify the home return speed (any positive number). Specify: `HomeReturnSpeed := value`
`MaximumSpeed`	Specify the maximum speed (any positive number). Specify: `MaximumSpeed := value`
`MaximumAcceleration`	Specify the maximum acceleration (any positive number). Specify: `MaximumAcceleration := value`
`MaximumDeceleration`	Specify the maximum deceleration (any positive number). Specify: `MaximumDeceleration := value`
`ProgrammedStopMode`	Enter Fast Stop, Fast Shutdown, or Hard Shutdown. Specify: `ProgrammedStopMode := text`
`AverageVelocityTimebase`	Specify the average velocity timebase (any positive number). Specify: `AverageVelocityTimebase := value`
`ServoStatusUpdateBits`	Specify the servo status update bits. Enter a hexadecimal number. Specify: `ServoStatusUpdateBits := 16#value`
`MotionConfigurationBits`	Specify the motion configuration bits. Enter a hexadecimal number. Specify: `MotionConfigurationBits := 16#value`
`AxisType`	Enter Unused, Position Only, Servo, Consumed, or Virtual. Specify: `AxisType := text`
`PositionUnwind`	Specify the unwind position (0-65,535). Specify: `PositionUnwind := value`
`MaximumPositiveTravel`	Specify the maximum positive travel (any positive number). Specify: `MaximumPositiveTravel := value`
`MaximumNegativeTravel`	Specify the maximum negative travel (any positive number). Specify: `MaximumNegativeTravel := value`
`PositionErrorTolerance`	Specify the position error tolerance (any positive number). Specify: `PositionErrorTolerance := value`

Attribute:	Description:
PositionLockTolerance	Specify the position local tolerance (any positive number). Specify: PositionLockTolerance := *value*
PositionProportionalGain	Specify position proportional gain (any positive number). Specify: PositionProportionalGain := *value*
PositionIntegralGain	Specify the position integral gain (any positive number). Specify: PositionIntegralGain := *value*
VelocityFeedforwardGain	Specify the velocity feedforward gain (any positive number). Specify: VelocityFeedforwardGain := *value*
AcclerationFeedforwardGain	Specify the acceleration feedforward gain (any positive number). Specify: AccelerationFeedforwardGain := *value*
VelocityProportionalGain	Specify the velocity proportional gain (any positive number). Specify: VelocityProportionalGain := *value*
VelocityIntegralGain	Specify velocity integral gain (any positive number). Specify: VelocityIntegralGain := *value*
OutputFilterBandwidth	Specify output filter bandwidth (any positive number). Specify: OutputFilterBandwidth := *value*
OutputScaling	Specify the output scaling (any positive number). Specify: OutputScaling := *value*
OutputLimit	Specify the output limit (any positive number). Specify: OutputLimit := *value*
OutputOffset	Specify output offset (any positive number). Specify: OutputOffset := *value*
FrictionCompensation	Specify friction compensation (any positive number). Specify: FrictionCompensation := *value*
SoftOvertravelFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: SoftOvertravelFaultAction := *text*
PositionErrorFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: PositionErrorFaultAction := *text*
EncoderLossFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: EncoderLossFaultAction := *text*
EncoderNoiseFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: EncoderNoiseFaultAction := *text*
DriveFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify: DriveFaultAction := *text*
ServoConfigurationBits	Specify the servo configuration bits. Enter a hexadecimal number. Specify: ServoConfigurationBits := 16#*value*
MotorEncoderTestIncrement	Specify the motor encoder test increment (any positive number). Specify: MotorEncoderTestIncrement := *value*
TuningTravelLimit	Specify the tuning travel limit (any positive number). Specify: TuningTravelLimit := *value*
TuningSpeed	Specify the tuning speed (any positive number). Specify: TuningSpeed := *value*

Attribute:	Description:
`DampingFactor`	Specify the damping factor (any positive number). Specify: `DampingFactor := value`
`PositionServoBandwidth`	Specify position servo bandwidth (any positive number). Specify: `PositionServoBandwidth := value`
`TuningConfigurationBits`	Specify the tuning configuration bits. Enter a hexadecimal number. Specify: `TuningConfigurationBits := 16#value`

Import/Export Version 1.1 RSLogix 5000 Version 8.0

Version 1.1 (major revision 1, minor revision 1) of the import/export utility that is included with RSLogix 5000 programming software, version 8.0 included these major enhancements:

- addition of function block instructions and routines
- addition of ASCII instructions
- verification of all instruction attributes and parameters

A

aliases 3-12, 6-4
array specifications 3-12
attributes
AXIS_CONSUMED TAG 3-16
AXIS_SERVO TAG 3-18
AXIS_SERVO_DRIVE TAG 3-22
AXIS_VIRTUAL TAG 3-17
CONTROLLER 2-5
DATATYPE 3-2
FBD_ROUTINE 3-34
MESSAGE TAG 3-14
MODULE 3-5
MOTION_GROUP TAG 3-14
PROGRAM 3-31
ROUTINE 3-33
TAG 3-13
TASK 3-35, 3-37
AXIS_CONSUMED TAG 3-16
AXIS_SERVO TAG 3-18
AXIS_SERVO_DRIVE TAG 3-22
AXIS_VIRTUAL TAG 3-17

B

backward compatibility B-1
BLOCK
component 5-9
guidelines 5-10
branches 4-2

C

comments 2-1, 3-29, 4-3, 6-2, 6-5
complete
branches 4-2
comments 2-1
components 2-2
CONFIG 3-36
connection list 3-7
CONTROLLER 2-4
conventions 2-1
DATATYPE 3-1
display style 2-3
FBD_ROUTINE 3-33
file format 3-1
function block logic 5-1
import/export 1-2, 1-3
MODULE 3-5
complete (continued)
PROGRAM 3-31
ROUTINE 3-33
rung logic 4-1
structure 2-2
TAG 3-10
TASK 3-34
components
basic format 2-2, 6-3
BLOCK 5-9
CONFIG 3-36
CONTROLLER 2-4
DATATYPE 3-1
descriptions 2-3
display style 2-3
FBD_ROUTINE 3-33
ICON 5-7
IREF 5-5
MODULE 3-5
OCON 5-7
OREF 5-5
PROGRAM 3-31
ROUTINE 3-33
TAG 3-10
TASK 3-34
WIRE 5-8
CONFIG
component 3-36
examples 3-41
connection list 3-7
CONTROLLER
attributes 2-5
component 2-4
example 2-6
guidelines 2-6
controller objects 3-36
conventions 2-1, 6-1
CSV format 6-1

D

DATATYPE
attributes 3-2
component 3-1
example 3-4
guidelines 3-4
descriptions 2-3
dimensions 3-12, 6-4
display style 2-3

E

examples
- ALIAS record 6-4
- COMMENT record 6-5
- CONFIG 3-41
- CONTROLLER 2-6
- DATATYPE 3-4
- function block logic 5-3
- ICON 5-8
- IREF 5-6
- MODULE 3-8
- OCON 5-8
- OREF 5-6
- partial project 6-7
- PROGRAM 3-32
- ROUTINE 4-2
- rung logic 4-3
- scenarios of partial import/export operations 6-6
- SHEET 5-3
- spreadsheet 6-8
- TAG 3-30
- TAG record 6-4
- TASK 3-36
- WIRE 5-9

Excel A-1

exporting
- .CSV format 1-6
- .L5K format 1-3
- complete project 1-3
- file structure 2-2
- partial project 1-6
- project 1-3
- tags 1-6
- types 1-1

F

FBD_ROUTINE
- attributes 3-34
- BLOCK logic 5-9
- component 3-33
- example 5-3
- ICON logic 5-7
- IREF logic 5-5
- OCON logic 5-7
- OREF logic 5-5
- SHEET logic 5-2
- WIRE logic 5-8

format
- CSV 6-1
- L5K 2-1, 3-1

function block logic 5-1

G

guidelines
- BLOCK logic 5-10
- CONTROLLER 2-6
- DATATYPE 3-4
- ICON logic 5-7
- IREF logic 5-6
- MODULE 3-8
- OCON 5-7
- OREF
  - logic 5-6
- PROGRAM 3-32
- rung logic 4-2
- SHEET logic 5-2
- TAG 3-30
- TASK 3-35
- WIRE logic 5-9

H

history, import/export utility B-1

I

ICON
- component 5-7
- example 5-8
- guidelines 5-7

importing
- .CSV format 1-4
- .L5K format 1-2
- complete project 1-2
- file structure 2-2
- partial project 1-4
- project 1-2
- tags 1-4
- types 1-1

initial values 3-28

instructions 4-3, 5-11

internal file comments 2-1, 6-2

IREF
- component 5-5
- example 5-6
- guidelines 5-6

L

L5K format 2-1, 3-1
logic 4-1, 5-1

M

MESSAGE TAG 3-14
Microsoft Excel A-1
MODULE
- attributes 3-5
- component 3-5
- connection list 3-7
- example 3-8
- guidelines 3-8

MOTION_GROUP TAG 3-14

N

neutral text 4-3, 5-11

O

objects 3-36
OCON
- component 5-7
- example 5-8
- guidelines 5-7

OREF
- component 5-5
- example 5-6
- guidelines 5-6

overview 2-2

P

partial
- ALIAS record 6-4
- COMMENT record 6-5
- comments 6-2
- components 6-3
- conventions 6-1
- CSV format 6-1
- dimensions 6-4
- example 6-7
- file format 6-1
- remark 6-2
- scenarios 6-6
- spreadsheet example 6-8
- TAG 6-2
- TAG record 6-3
- using Excel A-1

partial import/export 1-4, 1-6
PROGRAM
- attributes 3-31
- component 3-31
- example 3-32
- guidelines 3-32

projects 1-2, 1-3

R

remark 6-2
ROUTINE
- attributes 3-33
- component 3-33
- example 4-2

rung logic 4-1

S

SHEET
- example 5-3

structure 2-2

T

TAG
- aliases 3-12
- array specifications 3-12, 6-4
- attributes 3-13
- AXIS_CONSUMED 3-16
- AXIS_SERVO 3-18
- AXIS_SERVO_DRIVE 3-22
- AXIS_VIRTUAL 3-17
- component 3-10
- component comments 3-29
- example 3-30
- example (partial) 6-4
- examples 6-6
- guidelines 3-30
- initial values 3-28
- MESSAGE 3-14
- MOTION_GROUP 3-14
- partial 6-2
- record 6-3

tags 1-4, 1-6
TASK
- attributes 3-35, 3-37
- component 3-34
- example 3-36
- guidelines 3-35

text file 1-3

W

WIRE
component 5-8
example 5-9
guidelines 5-9

How Are We Doing?

Your comments on our technical publications will help us serve you better in the future. Thank you for taking the time to provide us feedback.

You can complete this form and mail it back to us, visit us online at www.ab.com/manuals, or email us at RADocumentComments@ra.rockwell.com

Pub. Title/Type Logix5000 Controllers Import/Export Reference Manual

Cat. No. Logix-based controllers Pub. No. 1756-RM084D-EN-P Pub. Date August 2001 Part No. 957603-58

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

Overall Usefulness	1 2 3	How can we make this publication more useful for you?
Completeness (all necessary information is provided)	1 2 3	Can we add more information to help you? ☐ procedure/step ☐ illustration ☐ feature ☐ example ☐ guideline ☐ other ☐ explanation ☐ definition
Technical Accuracy (all provided information is correct)	1 2 3	Can we be more accurate? ☐ text ☐ illustration
Clarity (all provided information is easy to understand)	1 2 3	How can we make things clearer?
Other Comments		You can add additional comments on the back of this form.

Your Name _____ Location/Phone _____

Your Title/Function _____

Would you like us to contact you regarding your comments?

___No, there is no need to contact me

___Yes, please call me

___Yes, please email me at _____

___Yes, please contact me via _____

Return this form to: Allen-Bradley Marketing Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705

Phone: 440-646-3176 Fax: 440-646-3525 Email: RADocumentComments@ra.rockwell.com

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE

Allen-Bradley
RELIANCE ELECTRIC
DODGE
ROCKWELL SOFTWARE
Rockwell Automation

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

Reach us now at www.rockwellautomation.com

Wherever you need us, Rockwell Automation brings together leading brands in industrial automation including Allen-Bradley controls, Reliance Electric power transmission products, Dodge mechanical power transmission components, and Rockwell Software. Rockwell Automation's unique, flexible approach to helping customers achieve a competitive advantage is supported by thousands of authorized partners, distributors and system integrators around the world.

Americas Headquarters, 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
European Headquarters SA/NV, avenue Herrmann Debroux, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-RM084D-EN-P - August 2001
PN 957603-58
Supersedes Publication 1756-RM084C-EN-P - June 2001

Copyright © 2001 Rockwell Automation. Printed in the U.S.A.